TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: B2612 – Elektrotechnika a informatika

Studijní obor: 2612R011– Elektronické informační a řídící systémy

Moderní multimediální frameworky

Modern multimedia framework

Bakalářská práce

Autor: **Jan Kovačka**

Vedoucí BP práce: Jiří Hnídek, Ing.

V Liberci 16.5.2008

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé BP a prohlašuji, že **souhlasím** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce.

Datum 16.5.2008

Podpis

Rád bych na tomto místě poděkoval vedoucímu práce ing. Jiřímu Hnídkovi za trpělivý dohled nad vznikem tohoto díla a za korekturu po věcné stránce.

Abstrakt

Bakalářská práce vysvětluje pojem multimediální framework a uvádí některé typy. Multimediální framework poskytuje programovatelné rozhraní pro práci s multimediálními soubory. Nejčastěji se jedná o uložení digitálního videa. Při zpracování digitálního videa multimediálním frameworkem záleží i na formátu, kterým bylo video zkomprimováno. Proto se práce věnuje i kompresi digitálního videa a vysvětluje některé principy a vlastnosti pocházející z analogového vysílání obrazu. Zaměňován bývá termín multimediální framework a multimediální kontejner a tak jsou vypsány základní formáty kontejnerů, aby se ucelila daná problematika.

Klíčová slova: multimediální framework, multimediální kontejner, komprimace videa, QuickTime, Matroska, H.264, VC-1

Abstract

The bachelor work explains term multimedia framework and presents some examples of modern types. Multimedia framework provides application programming interface for handling with multimedia files. Primary purpose is most often to store a digital video. Processing of digital video through multimedia framework depends on used type of video compression. That is the reason why is one part of the project devoted to digital video compression. Project explains basic principle of compression and features which come from analog television broadcasting. Frequently users exchange definition of multimedia framework with container. That is why I mention a list of common types of multimedia containers at the end.

Keywords: multimedia framework, container format, video compression, matroska, QuickTime, Matroska, H.264, VC-1

Obsah

Slovníček pojmů

Vysvětlení základních pojmů je důležité pro orientaci v celém obsahu práce. Množství informací si vynutilo zkrácení ustálených spojení do zkratek. Většina používaných technických výrazů má základ v anglickém jazyce, některé výrazy dokonce ani nemají český překlad a používají se v originále. Ve většině případů se zkratky také používají v anglickém jazyce, anglická terminologie přispívá k ujednocení pojmů, neboť pracovat se zkratkami českého překladu by bylo přinejmenším matoucí.

AAC

Celé označení zní MPEG-4 Advanced Audio Coding. Velmi kvalitní komprese zvuku umožňující práci s velkým množstvím kanálů. Tento kodek je patentován skupinou MPEG, tedy jeho komerční využití podléhá licenčním poplatkům. Do popředí se dostal především díky podpoře společností Apple Inc. a Nero AG. Společnost Apple Inc. si formát AAC upravila přidáním ochrany DRM (Digital Rights Management). Vedle klasického AAC - LC (Low-Complexity) pro bitrate větší než 80kbps existují ještě další profily AAC - HE (High Efficency) a rozšíření AAC - PS (AAC HE Parametric Stereo). AAC - HE je vhodný pro nízký bitrate pod 80kbps a používá techniku SBR (Spectral Band Replication), která dopočítává vysoké frekvence na základě analýzy nižších frekvencí. Profil AAC - PS je určen pro streamové přehrávání, protože dokáže zachovat kvalitní zvuk při 48kbps. AAC přehraje s patřičným pluginy většina dekodérů a dokonce je formát podporován i mobilními telefony.

AC3

Kompresní formát vyvinutý laboratořemi Dolby Labs v roce 1991. Setkáme se též s označení Dolby Digital (DD) nebo také SR*D. V začátcích byl formát projektován pro HDTV již v 80. letech, HDTV však bylo odloženo a tak se laboratoře snažily prosadit formát v kinech pod názvem SR*D (Spectral Recording Digital). Dolby Digital je vícekanálový zvukový systém 5.1 se vzorkovací frekvencí 32, 44.1 nebo 48kHz při 20 bitovém rozlišení. Datový tok je v závislosti na požadované kvalitě 320, 384, 448 nebo 640 kbps. Těmito parametry je předurčen pro záznam velmi kvalitního zvuku ve formátu DVD a HDTV. AC3 je většinou přehrávačů dobře podporován. Nejnovějším formátem je formát Dolby TrueHD určený pro 8 kanálů s maximálním datovým tokem 18 Mbps.

Aspect ratio

Neboli poměr stran. U obrazových snímků je dán poměrem horizontálního a vertikálního rozlišení (standardní poměr je 4:3 nebo 16:9). V některých případech může být poměr stran a rozlišení nezávislý, to v případě, že je poměr upraven softwarově. Pokročilé přehrávače umí vedle běžných poměrů 4:3, 16:9 i změnu na poměr 10:16, 1:2.35, 1:1.85 nebo 1:1.33). všechny tyto poměry jsou dány technologickými možnostmi zobrazovacích zařízení. Klasická obrazovka má poměr 4:3, širokoúhlá obrazovka je formátu 16:9 a displej širokoúhlého notebooku disponuje poměrem 16:10. Pokud se rozlišení změní vlivem změny poměru stran na jinou hodnotu, zachována musí zůstat dělitelnost 16 popřípadě 32 (např. rozlišení 720x544).

Blu-ray disk (častěji jen Blu-ray nebo BD)

Formát optických vysokokapacitních médií. V prvním čtvrtletí roku 2002 jej přivedla na trh firma Sony. Název vznikl díky barvě použitého laseru. Blu-ray technologie pracuje s laserovým paprskem o vlnové dělce 405nm, tato vlnová délka je v barevném spektru viditelného světla na rozhraní modré a fialové barvy (níže pod vlnovou délkou 400nm je již ultrafialové záření). Díky malým rozměrům čtecího a zapisovacího laseru je kapacita disku při dvouvrstvém záznamu 50GB. Krátce po uvedení Sony navrhla spolupráci i ostátním zájemcům o spolupráci, vznikla Blu-ray Disc Association (BDA) sdružující v dnešní době téměř 550 společností. Největším konkurentem byl, až do února 2008, formát HD-DVD.

Bit depth

Bitová hloubka neboli zvukové rozlišení audio signálu. Udává kolik různých hodnot může dosahovat jediný zvukový vzorek. Běžně se používá 16 bitů (65536 hodnot) u CD i DVD, pro kvalitní bezeztrátově komprimovaný zvuk formátu FLAC se používá 24 bitů.

Bitrate

Datový tok, který se udává nejčastěji v kbps (kilobits per second, kilobity za vteřinu). Obecně platí, že čím vyšší bitrate, tím lepší kvalita. Zde záleží ještě na rozlišení případně na použitém kodeku. Každý kodek má jiný optimální poměr bitů na pixel (MPEG-4 má poměr 0.2 – 0.5, MPEG-2 má poměr 0.3 – 1). Bitrate rozlišujeme na konstantní hodnotu (CBR, Constant bitrate) a na variabilní (VBR, Variable bitrate). VBR se průběžně mění tak, aby byla zachována kvalita i v datově náročných (rychle se měnících) scénách.

U nekomprimovaného zvuku při vzorkování 44.1 kHz a bitové hloubce 16bit je datový tok 1350 kbps. Pomocí komprese zvuku (MP3, Ogg Vorbis, MP4/AAC, MPC) dokážeme zvuk zkomprimovat aniž by si posluchač všiml rozdílu. Typická hodnota bitrate zkomprimovaného zvuku je 128 kbps, pro kvalitní zvuk (například zvuková stopa formátu AC3 pro video v HD kvalitě) je datový tok 640kbps.

DTS (Digital Theater System)

Formát původně navržený do kin. Zajímavostí je, že jedním z iniciátorů byl Steven Spielberg, který nebyl spokojen s existujícími zvukovými formáty (formát prvně použil pro film Jurský Park). DTS je taktéž systémem 5.1 a využívá kódování CAC (Coherent Acustic Codec), který umožňuje flexibilně měnit vzorkovací frekvenci až do 192 kHz. Zvýšené je i bitové rozlišení a to až na hodnotu 24bitů. Pro představu uvádím, že potřebný 24bitový převodník je v dnešní době stále velmi sofistikovanou technikou. DTS využívá konstantní bitrate 1183 kbps. Pochopitelně je vysoká kvalita záznamu vykoupena i větším objemem zvukové stopy a tak se s tímto formátem setkáme na vícevrstvých DVD a především pak na HD DVD a na discích BluRay.

Filtr

Používá se jako souhrnné označení pro nástroje na úpravu výstupu při přehrávání multimédií a pro splittery. Existují filtry na úpravu obrazu (práce s titulky, deinterlace, potlačení šumu atd.) a na úpravu zvuku

(normalizace zvuku, dynamická komprese, informace o zvukových stopách atd.). Filtry se používají pro úpravu výstupu nikoliv vstupu přehrávače.

FLAC (Free Lossless Audio Codec)

Bezeztrátový záznam zvuku. FLAC je šířen jako open source a je široce podporován. Parametry FLAC jsou obdobné jako u ostatních formátů zaměřených na kvalitní reprodukci, vzorkování je až 96kHz, bitová hloubka 16 bitů, bitrate stejný jako vstupní signál nebo poloviční.

GNU GPL (General Public License)

Projekt GNU se věnuje problematice svobodného softwaru. Zdrojové kódy pod licencí GPL mohou být svobodně upravovány a používány, šířeny však musí být opět pod licencí GPL a to obvykle bezplatně. Binární formy softwaru používající GPL mohou být poskytovány za libovolně vysokou úplatu. Ke GPL softwaru musí jeho autor/upravitel zdarma poskytnout zdrojové kódy.

GNU LGPL (Lesser General Public License)

Je varianta licence pro svobodný software GNU GPL, která ale na rozdíl od GPL umožňuje spojování s nesvobodným kódem.

HD-DVD (HighDefinition DVD)

Formát optických disků s kapacitou 30GB (pro dvouvrstvý záznam). Formát představila firma Toshiba krátce před uvedením konkurenčního formátu Blu-ray. Společnosti Sony (Blu-ray) a Toshiba (HD-DVD) tak započali „souboj formátů“, který trval 6 let a skončil "vítězně" pro firmu Sony. HD-DVD používá zápis a čtení modrým laserem o vlnové délce 405nm. Nižší kapacita disků oproti Blu-ray je dána nižší hustotou záznamu, HD-DVD má hustotu 8.8Gb na palec čtvereční což je o 6 Gb méně než zvládne zapsat konkurence. Společnost Toshiba vstoupila na trh jako jediný disponent své technologie. Po několika měsících přijmula firmu NEC za svého spolupracovníka. Oproti firmě Sony však vlivem licenční politiky sdružovalo fórum za podporu formátu HD-DVD jen pár set společností. To byl jeden z mnohých důvodů neúspěchu tohoto formátu.

Channels

Zvukové kanály. V praxi se vyskytují 3 hlavní konfigurace - mono, stereo a 5.1. Mono označuje zvuk s jedním kanálem, stereo má levý a pravý kanál, šestikanálový prostorový zvuk 5.1 značí pětici reproduktorů pro celé frekvenční pásmo a jeden reproduktor basový. Pro konfigurace obsahující více reproduktorů (rozložení 6.1 nebo 7.1) se další kanály nad konfirguraci 5.1 dopočítavají.

ISO (International Organization for Standardization)

Mezinárodní standardizační komise pro celé průmyslové odvětví. ISO sdružuje mnoho skupin a hierarchie je poměrně složitá. Pro nás bude podstatná pouze skupina MPEG.

ITU (International Telecommunication Union)

Mezinárodní telekomunikační úřad. Jeden z odborů, pod označením ITU-T, se zabývá doporučeními pro technické normy. Pro tuto publikaci je zajímavá pouze sekce doporučení pod písmenem H, ta se zabývá multimedii.

Kodek

Jedná se o český opis anglického „codec“. Jak v angličtině tak v češtině vznikl spojením slov KOder a DEKoder. Kodek slouží ke komprimování a dekódování jednotlivých streamů. Existuje spousta audio a video kodeků (k některým se dostaneme později). Někdy se za kodek nesprávně označují i filtry. Filtr je však nástroj pouze pro úpravu výstupního signálu, tedy pouze schopný přehrávat. K nejasnostem pak přispívá fakt, že s kodekem si uživatel nainstaluje do počítače zároveň různé filtry.

MP3

Celým označením MPEG-1 layer 3. Nejrozšířenější zvukový kompresní formát. Patent na tento formát vlastní firma Thomson/RCA. O vývoj se zasloužil německý vědec Karlheinz Brandenburg z pobočky Frauenhoferova ústavu pro mediální komunikaci. Existuje několik modifikací, ty se však neuchytili (MP3Pro, MP3 Surround). Bitrate zvukového stereo záznamu je 128 – 192kbps, bitrate může být i dynamický.

MPEG (Motion Picture Experts Group)

Skupina, která vznikla pod záštitou ISO. Je složena z vědecké i komerční komunity zabývající se vývojem průmyslových standardů pro zpracování videa.

Obrazové rozlišení (též jen rozlišení)

Určuje kolik bodů obrazu je v horizontálním a vertikálním směru. Obrazový bod se nazývá pixel. Pixel je pak nejmenší zobrazovací jednotka. Jeho relativní velikost je dána obrazovým rozlišením. Důležitější je absolutní velikost, která je dána technologickými možnostmi zobrazovací jednotky (monitor, projektor, atd.) Fotografická technika používá pro udání obrazového rozlišení jednotek Mpx (megapixel). Tuto jednotku dostaneme vynásobením horizontální a vertikální hodnoty rozlišení, udává kvalitativní hodnotu obraového rozlišení. Opět obecně platí, že čím vyšší rozlišení, tím kvalitnější je obraz.

Sampling Rate

Vzorkovací frekvence, s kterou je zachycena zvuková stopa. Udává, z kolika vzorků (samples) za vteřinu se skládá zvuk. Zvukové stopy na CD používají vzorkovaní o frekvenci 44.1 kHz, DVD 48 kHz. Vzorkovací frekvence je dvojnásobkem maximální slyšitelné frekvence (tj. 22kHz). Dvojnásobek maximální frekvence se volí záměrně, aby se splnil Shannonův teorém. Ten praví, že přesná rekonstrukce signálu z jeho vzorků je možná, pokud byl vzorkován frekvencí alespoň dvakrát vyšší než je maximální frekvence rekonstruovaného signálu.

Snímková frekvence

Vyjadřuje kolik je zobrazeno jednotlivých snímků za vteřinu. Označuje se zkratkou fps (Frames Per Second). V evropské normě je používáno 25fps. Číslo 25 je zvoleno s ohledem na citlivost oka a technologii zobrazování. Oko registruje plynulý pohyb při frekvenci 24 snímků za vteřinu, pod tuto hranici již obraz působí „trhaně“. Z technického hlediska je pak počet snímků 25 kvůli frekvenci síťového napětí, které má v Evropě hodnotu 50Hz. Zobrazování se děje v násobcích této frekvence a tak je pochopitelné proč se v Evropě používá 25fps a v Americe 29,97fps. Americká rozvodná síť pracuje na frekvenci 60Hz.

Splitter

Slouží přehrávačům pro rozdělení kontejneru na jednotlivé streamy, které poté předá patřičným dekodérům. Příkladem je Haali Media Splitter, který se používá pro přehrávání formátu MKV (MatroskaSplitter).

Stream

Je základní část multimediálního souboru. Stream (datový proud) může být video, zvuk, titulky nebo kapitoly. V jednom souboru může být i více streamů stejného typu, například několik jazykových verzí titulků a zvukových stop. Mohou být v různém formátu i kvalitě.

Vorbis

Formát kolem skupiny označené jako Xiph.org. Jedná se o zvukový kodek používaný v kontejneru OGG. Vzhledem k tomu, že je Vorbis open source, lze ho využít zdarma a to i pro komerční využití. Vorbis nabízí kvalitní zvuk při datovém toku od 64 do 320kbps a umí pracovat i s více kanály. Formát OGG je hojně podporován a přehrání zvuku OGG Vorbis zvládá většina dnešních přehrávačů.

WMA9

Formát Windows Media Audio společnosti Microsoft, který byl ustanoven spolu s video formátem WMV9 (Windows Media Video verze 9). Oproti MP3 má horší vlastnosti a srovnatelné kvality dosahuje především na spodních tónech. Stejně jako všechny kodeky od společnosti Microsoft, obsahuje WMA ochranu autorských dat DRM.

Úvod

Podstatou multimediálního frameworku (dále jen „framework“) je rámcování. Hierarchicky se rámcují příkazy pro práci s multimédii na základě tématické podobnosti. Podrobněji je téma popsáno ve třetí kapitole.

V práci nalezneme také kapitolu věnovanou kompresním formátům, neboť je důležité vědět s jakými daty framework pracuje. Na úplném začátku uvádím formáty televizního vysílání. I když by se zdálo, že s tématem frameworků televizní formáty nesouvisí, pomůže nám tato kapitola k objasnění a ucelení některých základních pojmů. Bez těchto informací by následující kapitola věnovaná kompresi obrazu nepřinesla srozumitelný výklad. Uvedením kompresních formátů se pak přímo dostaneme k problematice frameworků. V další kapitole se zaměřím na objasnění činnosti frameworků a to na konkrétních případech. Zmínku věnuji i multimediálním kontejnerům. Na závěr uvedu na jednoduché aplikaci praktické využití frameworku.

Úplně na začátek se pokusím definovat pojem multimediální framework:

Filozofie je založena na zapouzdření různých dat do obecného kompatibilního formátu, který bude co nejvíce nezávislý na přehrávané aplikaci a operačním systému. K přehrávanému médiu se přistupuje jako k obecnému rámci dat. Tento datový blok se pak dále analyzuje a extrahují se z něj zapouzdřená data. K těmto datům se teprve určuje, dle analyzovaného typu, vhodná „technologie“ přehrávání (přístupu).

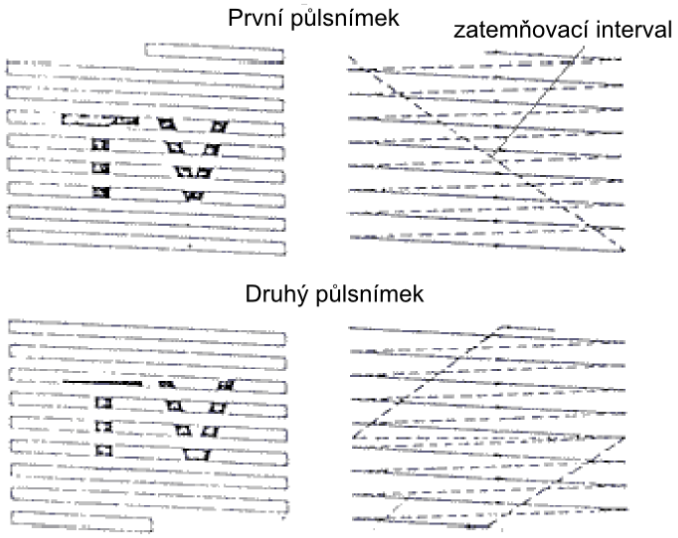
1. Televizní formáty

1.1 SDTV (Standard Definition TeleVision)

Televize běžného rozlišení (dále jen SDTV, běžné rozlišení bude dále v textu označeno jako SD). SDTV má poměrem stran obrazu 4:3 a rozlišením činí 720 x 576 při 25fps pro PAL a 720 x 480 při 29.97fps pro NTSC. Někdy je takovéto rozlišení označováno jako 480i, označení „i“ informuje, že jsou obrazové snímky prokládány (z anglického slova interlace).

1.2 Prokládání

Jedná se o způsob zobrazování v televizní technice, který znamená rozdělení jednoho snímku na sudé a liché řádky a přenos obrazu po takto vytvořených půlsnímcích. Vzniká tak dojem, že je vysíláno s frekvencí 50fps. Prokládání se typicky vyskytuje v souvislosti s TV přenosy, filmy točené do kina prokládání nemají.



Obr. 1-1 Princip prokládání a ukázka běhu světelného paprsku, [1]

Zobrazuje se bod po bodu, začíná se vlevo nahoře, pokračuje se doprava až ke kraji obrazu (konec řádku) a pak se postupuje směrem dolů. Na obr. 1-1 plné čáry určují běh paprsku, který rozsvěcuje jednotlivé pixely stínítka. Čárkované čary označují pohyb paprsku ve fázi, kdy je neaktivní. Jedná se o vertikální (mezi jednotlivými řádky) a horizontální (mezi půlsnímky) zatemňovací intervaly (blanking interval), které slouží k přesměrování paprsku na začátek řádku. U druhého půlsnímku se paprsek nevrací přímo, ale tzv. zig - zag.

1.3 Barevný televizní systém

V barevném systému se přenášejí tři základní barvy: červená (Red), zelená (Green), modrá (Blue) - zkráceně RGB. U všech složek se používá gama korekce pro redukci šumu při přenosu, hodnota například pro NTSC je 2.2 bodu. Kvůli zpětné kompatibilitě s černobílým signálem se přenáší zvlášť jasová a barevná složka. Kvůli citlivosti oka nelze barevné složky jednoduše sečíst, proto se používá vážený vzorec $Y = 0.299R + 0.587G + 0.114B$

1.4 Formáty televizního vysílání

NTSC (National Television System Committee) je formát televizního vysílání, který je standardem v USA, Japonsku a některých dalších zemích. Byl přijat v roce 1953, tedy o 14 let dříve než Evropský standard PAL. Šířka pásma monochromatické složky Y je 3.2MHz. Nad touto složkou je namodulována ještě barevná složka obsahující dvě části

$Q = 0.21R - 0.52G + 0.31B$

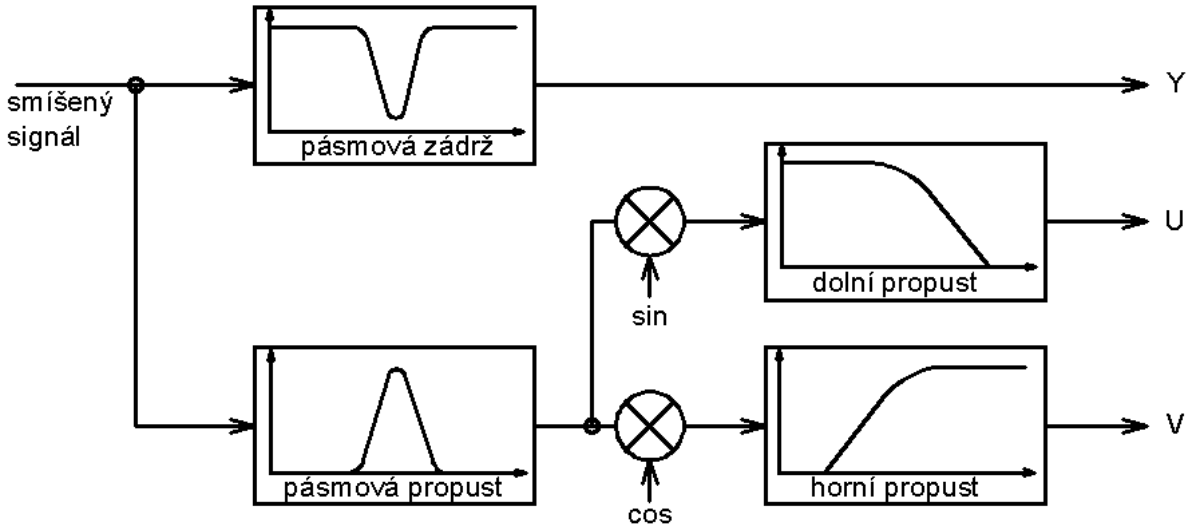
$I = 0.60R - 0.28G - 0.32B.$

Složky Q a I mohou nabývat kladné i záporné hodnoty. Pokud je Q kladné jde o purpurovou barvu, při záporné hodnotě o zelenou barvu. Kladná hodnota I vytváří oranžovou a záporná hodnota tyrkysovou barvu. Protože je oko citlivější na složku I, má šířku pásma 1.5MHz, složka Q pak jen 0.5MHz. Obě složky jsou kvadraturně namodulovány na nosnou frekvenci 3.58MHz, přičemž fáze složky I je stejná jako monochromatická a složka Q je o 90° posunuta. Polarita fáze je vyjádřena vzorcem $C = I \sin wt + Q \cos wt$.

PAL (Phase Alternation Line) je Evropský systém televizního vysílání. Používá podobný princip jako NTSC, nedefinuje ale pro přenášení barevný systém YIQ, nýbrž YUV. Složka Y zůstává stejná $Y = 0.299R + 0.587G + 0.114B$. Barevné složky pak získáme dle vzorců

$U = Cb = 0.492(B - Y)$

$V = Cr = 0.877(R - Y).$



Obr. 1-2 Princip oddělení složek YUV před převodem do číslicové podoby, [2]

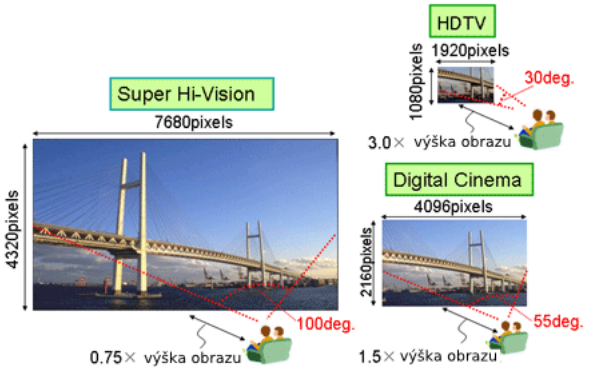
RGB hodnoty jsou opět gamma korigovány hodnotou 2.2 (i když ve standardu je napsána hodnota 2.8). U i V mají obě šířku pásma 1.3MHz a jsou namodulovány na frekvenci 4.43361875MHz. Každá druhá řádka obrací polaritu fáze V složky (odtud název Phase Alternation Line). V praxi to tedy znamená, že složky jsou popsány formulí $C = U \sin wt \pm V \cos wt$.

1.5 HDTV (High Definition TV)

Televize s vysokým rozlišením (dále jen HDTV, vysoké rozlišení bude dále v textu označeno jako HD). V 80. letech byl firmami Sony a NHK (Japonská rozhlasová společnost) vyvinut systém HDTV s názvem NHK Hi-Vision se stejným rozlišením jako profesionální 35mm film. Rozlišení HDTV videa je 1920x1080 nebo 1280x720 s poměrem stran 16:9. Vedle konkrétního rozlišení se dále profiluje, zda se jedná o prokládané či neprokládané snímkování. Přívlastek „i“ značí prokládání (interlace) snímků, označení „p“ znamená, že se obraz přenáší po celých snímcích. Zápis obrazového rozlišení se často zkracuje pouze na vertikální hodnotu, tzn. že rozlišení 1920x1080 bez prokládání snímků značíme 1080p. Rozlišení „plného“ HDTV (FullHD), jak je obchodně označováno rozlišení 1080p, činí zhruba 2Mpx (megapixely). To je důvod, proč je obraz ostřejší oproti SDTV, kde je rozlišení pouhých 0.35Mpx. Přenos HDTV je digitální. Vedle vysokého rozlišení a kvalitnějšího barevného podání přináší i kvalitní zvuk. Zvuk je šestikanálový v konfiguraci 5.1, což znamená, že je zprostředkován třemi reproduktory vpředu, dvěmi za zády diváka a jedním přídavným basovým reproduktorem. Běžně se používá zvuková komprese pomocí kodeku AAC (MPEG-4 Advanced Audio Coding) nebo AC3 (formát Dolby Digital). Pokud je záznam pořizován na vysokokapacitní disky používá se i bezeztrátová komprese zvuku.

Další vývoj se ubírá v duchu „High Definition“, jde tedy o zvyšování rozlišení. Po formátu HDTV následuje označení XD (eXtreme Definition) mající rozlišení 2560 x 1440. Takovéto rozlišení najdeme u kvalitních profesionálních 30“ (třiceti palcových) LCD monitorů. Rozlišení SHD (Super High Definition) nejdeme například u nejvýkonnější karty Matrox Parhelia, rozlišení SHD je 3840 x 2048.

Nejnovějším počinem z laboratoře NHK je standard UHD (Ultra High Definition) označován jako Super Hi-Vision. Rozlišení je úctyhodných 7680 x 4320. Obrázek ilustruje pozorovací úhly pro formát HDTV, D-Cinema (Dolby Digital Cinema – digitální formát pro reprodukci v kinech) a UHD při doporučené pozorovací vzdálenosti.



Obr. 1-3 Porovnání pozorovacího úhlu z relativně stejné vzdálenosti od obrazu,[3]

2. Komprese digitálního videa

Přesun ke kvalitnějšímu zobrazení v HD kvalitě a vysoké rozlišení u mobilních zařízení, vede k vysokým nárokům na přenos videa. Technologie přenosových tras jsou však limitovány a narůstající potřeba přenášet stále větší objemy dat, při stejné přenosové šířce, vede nutně ke kompresi dat. Z původního standardu MPEG-2 (popř. MPEG-4) se do dnešní doby vyvinulo mnoho různých modifikací a principů. Díky pomalé standardizaci v 90. letech se zrodily dva základní formáty významné pro další vývoj – byly to formáty komprese H.264 a VC-1.

Motivací pro vývoj bylo zřizování paralelních služeb na jedné datové trase, snižování nákladů, HD televize a nutnost efektivnější archivace. Vývoji přispěla i silná podpora ze strany výrobců hardwaru, zvláště pak v odvětví přenosných technologií. Nejvýraznějším motivačním prvkem ve vývoji komprimačních standardů digitálního obrazu byl přenos HDTV. Oproti běžnému video signálu (SDTV) vyžaduje díky vysokému rozlišení (HDTV) zhruba pětkrát větší datový tok.

2.1 Historie kompresních formátů

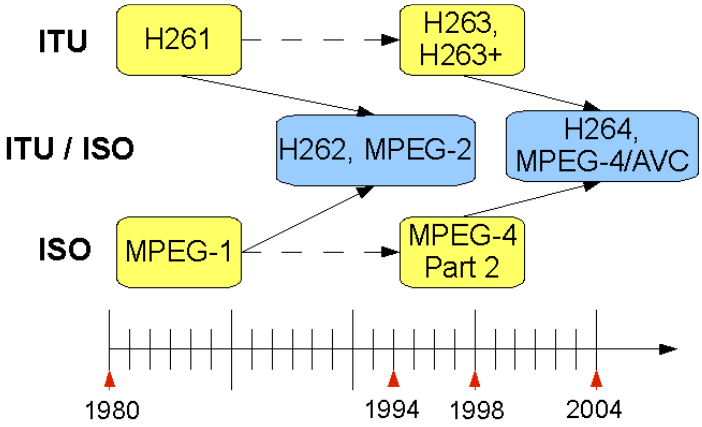
V polovině osmdesátých let vznikla řada speciálních video kodeků (CODECS). Zejména pro potřebu PC vznikla řada proprietálních formátů jako AVI, Cinepak a QuickTime.

Narůstající počet kodeků vedl na konci osmdesátých let k vytvoření ISO standardu. Vznikl MPEG-1 určený pro digitalizaci na CD. Jelikož však tento standard nevyhovoval pro přenos dat, byl vytvořen paralelní standard pro telekomunikační potřeby ITU H.261. Začátek devadesátých let znamenal nutnost inovace standardu MPEG-1 tak, aby byl vhodný i pro přenos po datových trasách. Tím se objevil na scéně MPEG-2 (zpětně kompatibilní s verzí 1). Skupina ITU vydala nový standard H.26, ten již byl zahrnut i do MPEG-2. V této době vznikla myšlenka na televizní obraz ve vysokém rozlišení – vznikla myšlenka a pojem HDTV. To odstartovalo vlnu nových metod v oblasti komprimace dat. Jeden z prvních formátů měl být MPEG-3. Jeho koncepce byla však do velké míry obsažena již ve standardu MPEG-2 a tak byl vývoj tímto směrem zrušen.

Druhá polovina devadesátých let způsobila nárůst především bezdrátových technologií. Přenos na úzkých pásmech dal vzniknout formátu ISO MPEG-4, ITU H.263 a H.26L. Vznik však provázely boje o patenty a následné licenční poplatky. Tím vznikly další interní formáty RealMedia, Windows Media a QuickTime. Jako nejvhodnější formáty i pro přenos se nakonec ukázaly MPEG-4 a H.263, čímž vytvořily konkurenci pro MPEG-2.

Došlo ke konvergenci směrem ke kvalitní kompresi (MPEG-4 převedl MPEG-2 z DVD na CD). Došlo také k první „inspiraci“ pro další formát. Vznikl projekt DivX, který se ze začátku vyvíjel z MPEG-4, později kvůli patentním sporům se přiklonili autoři k H.263.

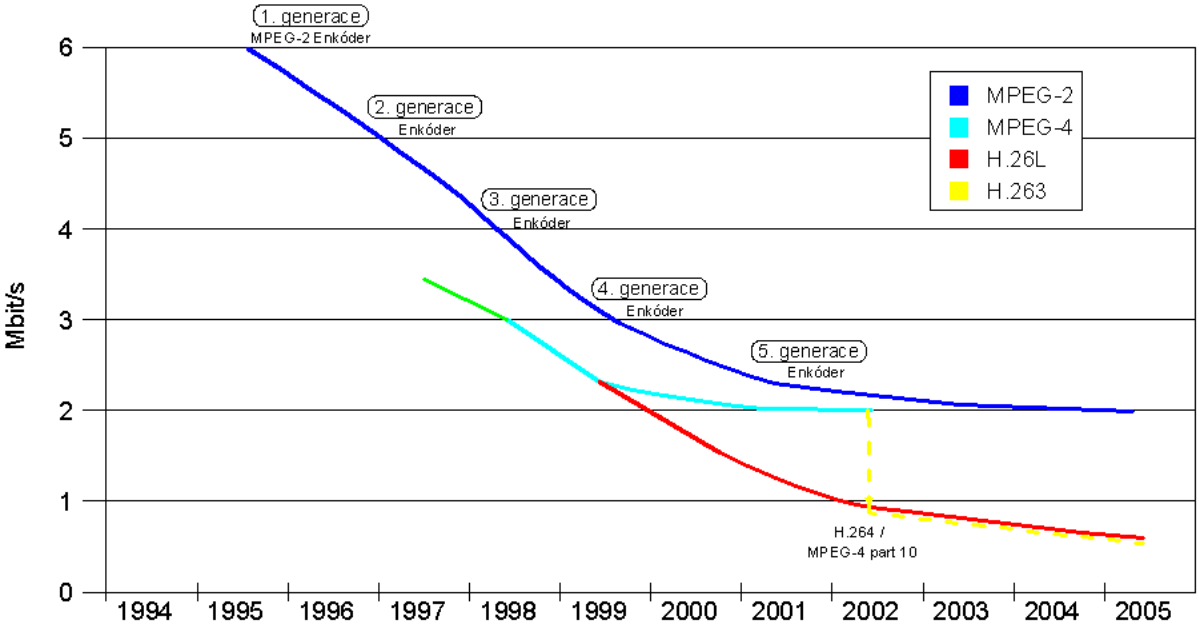
Hrozil neúspěch formátu MPEG-4, jakožto komplexního formátu a tak



Obr. 2-1 Vývoj standardů ITU/ISO na časové ose, [4]

vznikl výňatek týkající se jen komprese videa a tato podmnožina se spojila s ITU H.264. H.264 vznikl opět paralelně jako odezva na inovace ve formátu MPEG-4. Spojením vznikl nový profil H.264/MPEG-4 AVC (jinak označován také pouhým AVC, Advanced Video Coding). Tímto formátem se ustanovily některé nejasnosti ve formátu MPEG-4, pravidla kódování a způsob přehrávání. Spojením ISO a ITU vznikl kvalitní konkurenceschopný formát a celkově se zvedla kvalita komprimace.

Microsoft mezitím vyvíjel svůj vlastní formát VC-1. Z počátku šlo pouze o interní výtvor pro Windows Media verze 9 (později pro WM 10). Obecný vnitřní kodek VC-1 Microsoft prosadil i jako alternativu pro off-line záznam HDTV. WMV9 navíc implementuje i interaktivní vlastnosti a systém ochrany autorských dat DRM. Přestože existuje značná podobnost mezi AVC a VC-1, nejsou spolu kompatibilní.



Obr. 2-2 Vývoj generací enkóderů a jejich dosažený datový tok, [4]

Geneze formátů a standardů vedla k efektivnějšímu dekódování a zmenšení datového toku. Po formátu MPEG-4 vznikl i další formát MPEG-7. Ten se však více než stránkou komprimace videa zabýval standardizací nástrojů, pravidly indexace, organizací a metodikou vyhledávání.

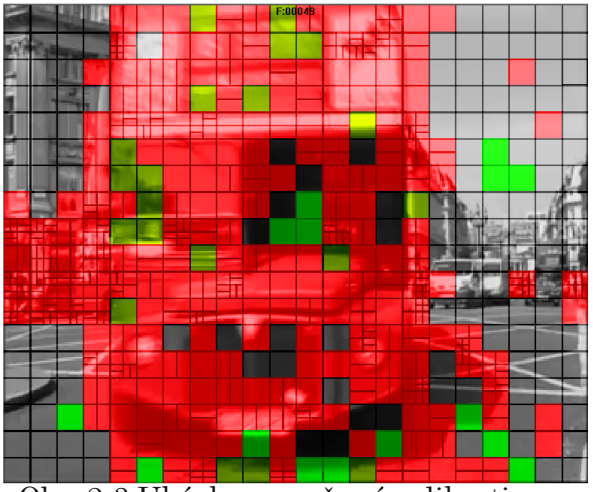
2.2 Měřítko kvality komprese

Měřítkem je, kolik bitů musíme mít pro přenos videa, aby zůstala zachována stejná vizuální kvalita. Vizuální kvalita je však velmi subjektivní pojem a závisí na mnoha faktorech. Prakticky lze provést téměř výhradně jen pozorovatelem, automatické vyhodnocení je zatím ve vývoji. Testování se nejčastěji děje metodou PSNR (Peak Signal-to-Noise Ratio), což znamená pozorování na úrovni perceptuálně významného šumu. První tři generace formátu MPEG-2 se liší takřka jen v tomto ohledu.

2.3 Základní principy komprese

Nejprve je potřeba ujasnit, že bezeztrátový přenos není možný (jen SD obraz má datový tok 12Mbps). Zabýváme se tedy vždy jen ztrátovou kompresí, ať už je výsledný soubor jakkoliv veliký. Základní princip spočívá v rozdělení snímku na makrobloky. Makroblok je skupina pixelů, která se zpracovává nezávisle.

Makroblok má (ve formátech ISO MPEG-1 a MPEG-2) konstantní velikost 16 x 16. Konstantní dělení nevystihuje nerovnoměrné rozložení detailů obrazu. Překonání tohoto problému přináší standard MPEG-4 s adaptivním dělením makrobloků (lepší přizpůsobení detailům obrazu). Pro každý makroblok se může uskutečnit jiné kompresní zpracování. Pokud však navazující makrobloky mají odlišný kom-presní poměr, dochází k tzv. Blocking Artefact, to jest zviditelnění hranic mezi jednotlivými makrobloky. MPEG-4 a VC-1 zavádějí pro přehrávače sadu deblocking filtrů. Enkódery pak mohou dokonce přehrávači „napovědět“ jaký filtr bude nejvhodnější použít.



Obr. 2-3 Ukázka proměnné velikosti makrobloků

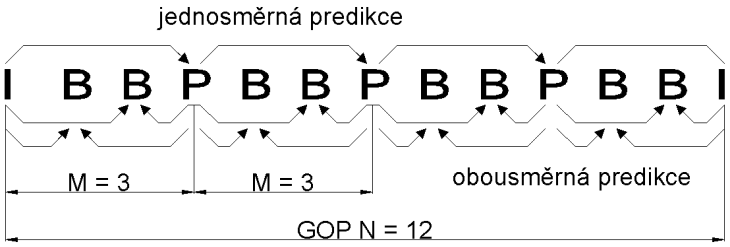
Obr. 2-4 Ukázka pohybových vektorů

Při kompresi se dále využívá časoprostorové korespondence bloků, kdy se určitá část obrazu nemění. Tento princip využívají IBP bloky. Význam jednotlivých bloků je následující – blok I (inerframe) označuje referenční blok, B je blok mezi I a P, konečně blok P (prediction) předpovídá další vývoj detailů v obraze. Hledání korespondence je pochopitelně nejednoznačnou záležitostí a vyžaduje velmi náročné metody analýzy obrazu. V původních ISO standardech se IBP objevuje taktéž, ale na úrovni obrazů (IBP frames).

Nejefektivnější způsob je ten, kdy ke korespondenci pomocí IBP bloků přidáme směr pohybu, tzv. Motion Vector. Motion Vektor umožňuje efektivní kompresi makrobloků v rámci sekvence.

Na následující ukázce je zobrazena skupina obrázků (GOP, Group Of Pictures) tak jak je předepsáno pro formát MPEG-2. Velikost ohraničují snímky I.

Metody komprese dělí každé obrazové okénko na nezávislé oblasti – řezy (slices). Řezy lépe rozdělují pasivní a aktivní část obrazu. V kombinaci s adaptivními makrobloky a vektory pohybu se dosahuje vysoké komprese. Řezy mohou být nezávislé v prostoru a tím se docílí lepší kódování pohybujících se detailů v obraze.



Obr. 2-5 – Ukázka řazení snímků IBP v GOP, délka skupiny N, periodicita M, [6]

Vlastní ztrátová komprese se provádí na úrovni každého makrobloku. Bloky se převádějí do matematického prostoru koeficientů příslušné transformace. MPEG-1 a MPEG-2 využívají striktně koeficienty DCT (Discrete Cosine Transform) neboli převod do frekvenční oblasti. DCT je podobná Fourierově transformaci a dopředný tvar zapíšeme tako:

$$S(v,u)=\frac{C(v)C(u)}{4}\sum_{y=0}^{7}\sum_{x=0}^{7}\left[s(y,x)\cos\left[\frac{(2x+1)u\Pi}{16}\right]\cos\left[\frac{(2y+1)v\Pi}{16}\right]\right]$$

Koeficienty se následně kvantují, tedy se převádějí z desetinné plovoucí čárky do celočíselné podoby. Tímto se dostáváme k nejdůležitější části komprese z hlediska ztráty informace, která se děje zaokrouhlováním a vážením celočíselných koeficientů. Sofistikovanější moderní metody využívají diskrétní matematické transformace přes celočíselné maticové operace. Dochází tak k menším ztrátám a snáze se implementuje. Výsledný bitový tok se nakonec bitově kóduje. MPEG-2 tak činí entropickým Huffman kódováním, jehož základní myšlenkou je zakódování znaků podle počtu jejich výskytu na vstupu. V případě H.264 se jedná o efektivnější způsob zakódování metodou CABAC (Context Adaptive Binary Arithmetic Coding).

110	106	98	92
112	105	97	90
109	107	97	94
111	107	95	91

a) obrazové prvky

203	15	0	-1
0	0	0	1
0	0	0	0
1	-2	0	-1

b) koeficienty DCT (po zaokrouhlování)

Obr. 2-6 Transformace DCT pro matici 4x4, [6]

2.4 Podrobněji o základních standardech

TERMINOLOGIE MPEG-4

Původní standard MPEG-4 byl vytvořen jako obecný a velmi komplexní formát. Soubor doporučení a obecných rozšíření vedl k enormním nárokům na finální implementaci kóderů a dekóderů v praxi. Ve standardu ISO 14496 (shrnutá norma pro MPEG-4) se objevilo mnoho chyb a nejasností. Byla nutná revize této normy a tak vznikl upravený výňatek ISO 14496-2 (označován také jako MPEG-4 part 2 Visual). Obsahuje definice kódovacích a dekódovacích profilů různých úrovní a rozlišení, vhodných pro použití v průmyslových aplikacích. Přesto, že se jednalo jen o část původního MPEG-4, šlo stále o příliš složitou implementaci. Ve stejné době vznikl ITU H.26L. Krok vpřed ve vývoji znamenala spolupráce obou paralelních větví, spojila se ITU a ISO MPEG. Vznikl tak nový formát H.264/AVC. V MPEG terminologii označován jako MPEG-4 part 10, v ISO terminologii jako ISO/IEC 14496-10 a ITU používal název H.264.

Největšího konkurenta pro formát H.264/AVC vytvořil Microsoft v podobě VC-1 (interní formát WMV9). V roce 2003 byl standardizován jako SMPTE VC-1. Z původního záměru firmy Microsoft, aby VC-1 sloužil jen jako interní kodek, se vlivem popularity mezi uživateli nakonec prosadil jako komerční a průmyslově využívaný standard i pro HD obraz a jeho reprezentaci na HD-DVD disky.

MPEG-4 PART 2

Jedním z důvodů neúspěšnosti tohoto formátu je paradoxně myšlenka dokonalejší komprimace, ta hovoří o systému komprimace vstupu. Předpokladem je, donutit vstupní zařízení „přemýšlet“ nad správným dělením obsahu. Jako příklad vezměme rozdíl mezi audio formáty MIDI a WAV. Pokud budeme přenášet zvlášť jednotlivé nástroje a noty, budeme používat standardní systém přehrávání a sadu nástrojů spolu se systémem cache, dostaneme velmi efektivní kompresi zvuku. Na tomto principu pracují polyfonní melodie některých mobilních telefonů (výrobky společností, které se rozhodly licencovat MPEG-4). V obrazové podobě si myšlenku můžeme ukázat

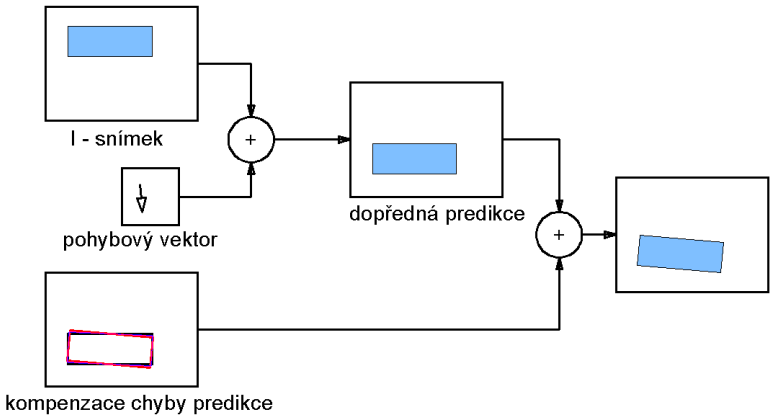
na příkladu malovaného filmu. Pozadí ve scéně se často nemění (je statické) a fáze aktérů jsou omezené. Pokud bychom měli standardní oddělený způsob přenosu pozadí, postaviček, jejich fází a pohybu dostaneme opět kvalitnější komprimaci. Avšak standardní přehrávání by při tomto způsobu kladlo vysoké nároky na přehrávač a byl by problém dodržet jednotnost výstupu.

Princip rozdělení do vrstev uplatněný v rámci MPEG-4 je označován jako systém kódování „vícevrstvých kompozic" tzv. Sprites and Layers. Jednotlivé objekty (sprites) jsou komprimovány samostatně a to včetně masky, pohyb je zakódován do proudových dat (sprite streaming) a scéna je hierarchicky složitě reprezentována. Je myšleno i na objekty měnící tvar, pro které je zavedena polygonální reprezentace 2D objektů (Mesh definition of Object), schopna efektivně zakódovat i 2D morphing. Pro reprezentaci 3D je dokonce zvolen speciální jazyk na principu VRML (Virtual Reality Modelling Language). Při realizaci těchto způsobů komprimace by zdrojový kóder musel buď skvěle segmentovat nebo mít přístup k samotným vstupním kanálům. V průmyslovém nasazení je tato myšlenka nerealizovatelná neboť se přehrávač mění na „animační nástroj" (tím pádem kladeny vysoké nároky na procesor a paměť). Jistým řešením by byla profilová specializace, ale tím by se popřela schopnost obecného přístupu k datům.

H.264/AVC

Tento standard opustil myšlenku zasahování do vstupní části a soustředil se jen na efektivní zakódování obecného vstupního signálu bez přehnaných nároků na dekódování. Prakticky se odstraňují vrstvy, objekty apod., které zaváděl původní koncept MPEG-4. Nový formát se navrátil k osvědčeným metodám a snažil se o jejich vylepšení. Obraz se dělí na makro-bloky s adaptivním rozdělením, jejich velikost variuje od 16x16 po 4x4. Dělení obrazového snímku je dynamické, tedy nemusí se nutně jednat o čtvercové rozdělení. Místo vrstev se zavádí pojem řezy (slices), což jsou obdélníkové oblasti se zarovnáním na 16 pixelů. Tímto způsobem je možné rozdělit scénu na vzájemně navazující bloky. Pro makrobloky je zavedena interpolace IBP povahy, podobně jako u MPEG-2.

Kódování makrobloků probíhá i v rámci jednoho snímku (Intra-Frame Coded Macroblocks). Idea je taková, že v rámci obrázku jsou opakující se motivy, které lze odvodit od předchozího makrobloku nebo jejich řady. Princip kódování pohybu je uplatněn i v rámci jednoho obrázku. Vhodně zvolíme počáteční makroblok, naznačíme predikci pohybu ostatních a využijeme k tomu IBP kódování. Dále je používán princip více referenčních obrázků (Multiple Arbitary Reference Frames) pro predikci bloků v rámci sekvence. V H.264/AVC se opustilo striktní dodržování predikce pomocí jednoho referenčního snímku (u formátu H.263 jednoho předchozího ref. snímku, u formátu MPEG-2 jednoho předchozího a následujícího ref. snímku), místo toho je reference libovolně volena pomocí až pěti předchozích a pěti následujících snímků. To lze využít pro různé prolínací efekty, zvolíme počáteční a koncový snímek a snímky mezi nimi dostaneme volbou váženého mediánu P a B snímků (Bi-Predictive Mode).



Obr. 2-7 Kompenzace chyby, která vzniká odhadem vektoru pohybu, [7]

Vylepšení kompenzace pohybu (Motion Compensation) se snaží o efektivní predikci a zakódování pohybu makrobloků. Přesnost je možná až na ¼ pixelu. Tato subpixelová přesnost je důležitá, neboť potlačuje vlastní blokové artefakty a vystihuje lépe pomalé pohyby v horizontálním směru (na horizontální pohyb je oko citlivější). Zavedením více druhů vektorů pohybu, lze dosáhnout standardního řazení vlastních vektorů pohybu a tím vytvářet příslušné skupiny vektorů pohybu a korigovat jejich odhad.

Nalezení vektoru pohybu je velmi obtížné a nejednoznačné, jeden pohyb lze zakódovat více způsoby. Hledáme vždy ten nejlepší pro kompresi, tj. ten co má nejvíce souhlasných směrů ve skupině. Tento fakt je podpořen variabilitou makrobloků a jejich typy. Vylepšilo se i vlastní kódování DCT transformace. Místo původního zpracování v plovoucí desetinné čárce, se přešlo na speciální celočíselnou variantu převodu. Ta je rychlejší a snadno ji lze implementovat.

Z profesionálního průmyslového hlediska je důležitá implementace kódování prokládaného signálu. U televizního signálu se prokládané snímkování využívá jako přirozená komprese přenášeného obrazu. Standardy SDTV i HDTV podporují prokládání a tak je nutné, aby bylo podporováno i kompresními formáty jakým je například H.264/AVC. Nelze totiž pracovat v prokládaném režimu, jako by se jednalo o jednotlivé snímky, protože by se projevily kompresní artefakty v časově-prostorovém rámci, takže se zpracovávají dva půlsnímky samostatně. H.264/AVC používá pro kódování půlsnímků dva principy. PAFF (Picture Adaptive Frame Field) kombinuje oba půlsnímky nebo kóduje každý zvlášť, tato metoda je vhodná zejména pro statické scény. MBAFF (Macroblock Adaptive Frame/Field) používá metodu PAFF v rámci samostatných navazujících makrobloků. To znamená, že kóduje dvojice makrobloků jako celky nebo samostatně po půlsnímcích.

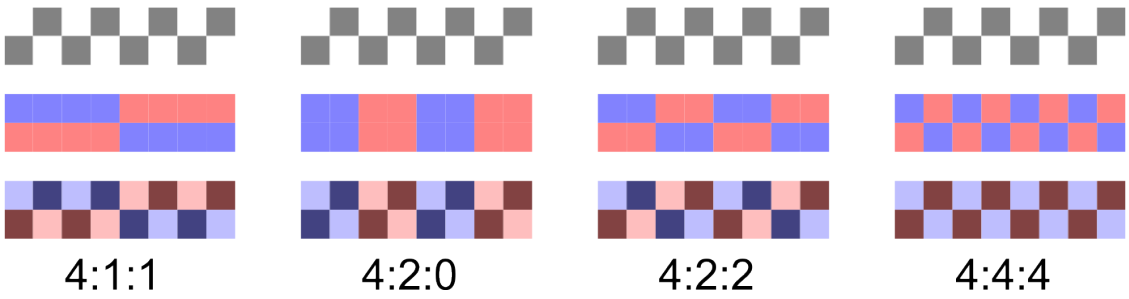
Výsledný proud koeficientů je třeba ještě nakonec bezeztrátově zkomprimovat, MPEG-2 používá Huffmanovo kódování (LZH bezztrátová komprese - podle autorů Lempel-Ziv a Haruyasu, základem metody je Huffmanovo kódování, s tím, že se pracuje s posuvným okénkem). H.264/AVC používá CABAC (Context Adaptive Binary Aritmetic Coding), což je bezztrátové kódování navržené přímo pro proud koeficientů DCT. Princip vychází z více než dvaceti modelů makrobloku a k nim 399-ti předdefinovaných kontextových modelů používaných ve skupinách.

Pro zjednodušení práce kóderů a dekóderů byly vytvořeny čtyři základní profily – Baseline (zejména pro D-Cinema), Main (pro většinu aplikací; podporuje B-slices, CABAC), Profile X (určeno pro streaming) a FRext (pro záznam HDTV). Jednotlivé profily pak mají ještě dalších jedenáct úrovní.

Microsoft VC-1

Na vzniku formátu se podílela řada odborníků, kteří přešli pod křídla firmy Microsoft právě z vývojových skupin ITU a ISO. Není tedy divu, že mají VC-1 a H.264 spoustu společného a používají stejné principy. V komerční implementaci nalezneme variabilní dělení makrobloků, využívání IBP principu a čistě celočíselnou šestnácti bitovou aritmetiku (bez plovoucí čárky je menší zatížení CPU). Výpočet je prováděn SIMD (Simple Instruction Multiple Data), principiálně umí paralelní výpočty, které lze uplatnit u více procesorových systémů. Důležitou vlastností jsou intra-predictive vektory, které přímo predikují DCT koeficienty. VC-1 má také speciální podporu interpolace snímkování (například z 15fps na 25fps).

Princip kódování je u formátu VC-1 i H.264/AVC velmi podobný, podstatný rozdíl je v systému predikce makrobloků. H.264/AVC predikuje makrobloky v oblasti obrazových bodů. VC-1 převede makrobloky na koeficienty upravené DCT transformace a predikuje na úrovni transformovaných koeficientů, tento způsob vede na velmi efektivní způsob kódování. Jedná se tedy o kódování ve frekvenční oblasti (je to výsledek výzkumu v rámci „vlnek“, tzv. Wavelets and Filterbanks), které je známé i ve skupině MPEG a to u formátu JPEG2000. Výhoda interpolace a predikce ve frekvenční oblasti je, mimo jiné, i výrazné odbourání blokových artefaktů a urychlení výpočtu při menší paměťové náročnosti. Zvláštním důsledkem je například možnost interpolace snímků v rámci času (již zmiňovaný převod 15fps na 25fps nebo odstraňování prokladu snímků).



Obr. 2-8 Ukázka barevného převzorkování, [8]

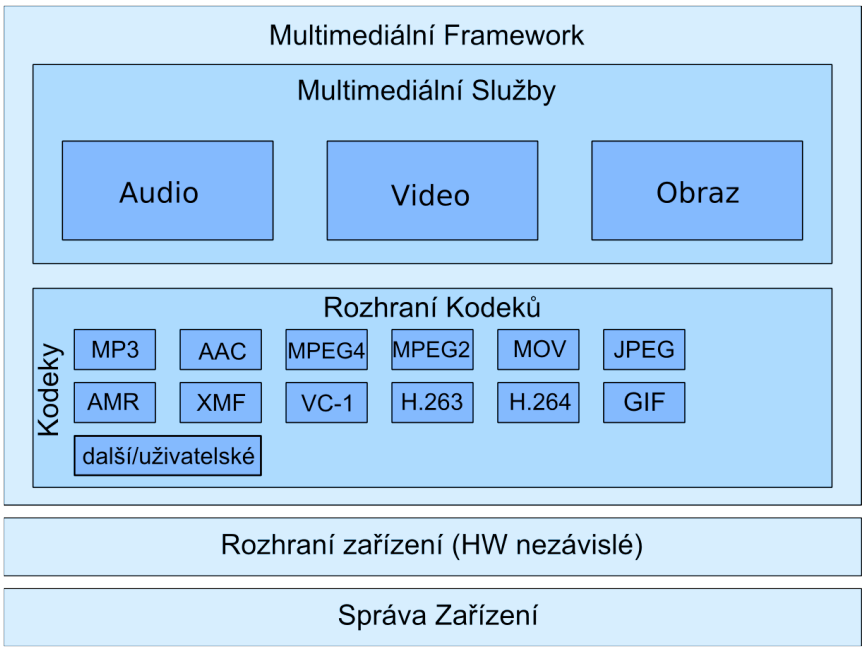
Mezi dalšími rozdíly těchto dvou formátů nalezneme jiný, osmi bitový, formát VC-1 a barevný prostor YUV s kódováním 4:2:0 (H.264/AVC obsahuje více možností kódování, včetně vzorkování 4:2:2 a 4:1:1). VC-1 má dále možnost implementování autorské ochrany DRM a dále využívá méně procesorového výkonu.

3. Moderní multimediální framework

Multimediální framework je softwarová struktura (obvykle soubor knihoven), která obsluhuje různé druhy multimediálních souborů na počítači prostřednictvím jediného rozhraní. Dobrý multimediální framework nabízí modulární architekturu pro snadnou podporu nových kodeků a intuitivní API (Application Programming Interface, programovatelné rozhraní aplikací).

Framework znamená rámcování a v případě multimediálního frameworku se jedná o rámcování příkazů pro ovládání multimédií. Jednoduché příkazy jsou hierarchicky sdružovány nadřazeným rámcem, rámce jsou seskupeny další nadřazenou službou a nakonec máme vše k dispozici skrze jedno rozhraní.

Obecné schéma popisující strukturu frameworku nelezneme na obr. 3-1. Požadavek na multiplatformní použití zajišťuje oddělení frameworku od rozhraní pro ovládání zařízení. Požadavek na modulárnost splňuje rozhraní kodeků a požadavek na obecný přístup k datům je dán oddělením služby pro video, audio a text.



Obr. 3-1 Obecné schéma multimediálního frameworku, [8]

Nejlépe bude začít s konkrétním multimediálním frameworkem. Jako vzorový jsem vybral QuickTime společnosti Apple Computer, Inc. (dále jen „Apple“). Hlavním kritériem pro výběr zmíněného frameworku je srozumitelnost a kvalitní dokumentace. Z multimediálních frameworků je QuickTime jedním z nejrozšířenějších a je v podvědomí širší počítačové veřejnosti. Často je spojován jen s designově zvládnutým přehrávačem a málo uživatelů tuší, že jde o komplexnější aplikaci.

Seznam nejběžnějších multimediálních frameworků je rozdělen podle toho, na kterou platformu je framework primárně určen. Většina z nich se snaží být nezávislá na platformě, příkladem jsou uvedené frameworky pro systém Linux, které najdeme i v systému Windows.

Linux a nezávislé platformy:

Helix DNA
GStreamer
FFmpeg
Xine

Microsoft Windows:

QuickTime
DirectShow
DirectX Media Objevte (DMOs)
Media Foundation (pouze Windows Vista)
Video for Windows (WwW) (známé jako Video Compression Manager (VCM))

Apple Mac OS:

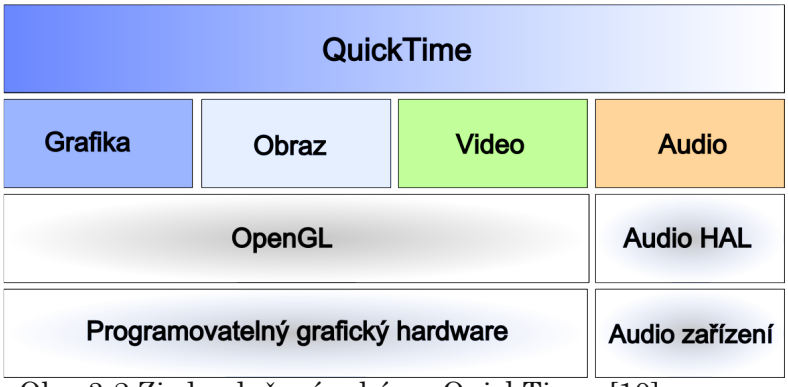
QuickTime

Symbian:

MMF (Multi Media Framework)

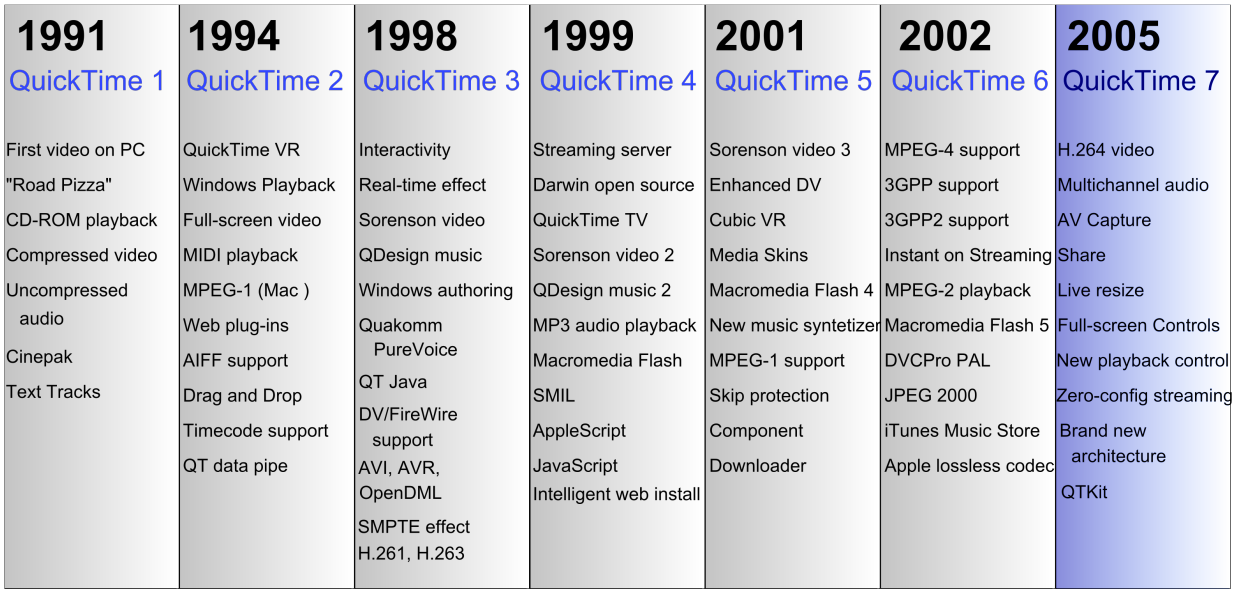
3.1 QuickTime

QuickTime (dále jen QT) je multiplatformní multimediální architektura původně určena pro Mac OS. Rozšířen je i na platformu Windows, na které bude dále popisován. QT je složen ze soustavy multimediálních systémových rozšíření (implementovány jako knihovny DLL), komplexní API (Application Programming Interface, programové rozhraní aplikace, dále jen API), souborového formátu a soustavy uživatelských aplikací jako je přehrávač QuickTime, ovládací prvky ActiveX (pouze platforma Windows) a plug-in webového prohlížeče.



Obr. 3-2 Zjednodušené schéma QuickTime, [10]

Díky dlouhodobému vývoji je QT propracovaný a především stabilní multimediální nástroj. Apple je úspěšná firma v počítačové oblasti a tak není divu, že drží krok s nejnovějšími technologiemi.



Obr. 3-3 Vývoj podporovaných formátů QuickTime, [9]

QT je kompletní multimediální architektura, nejen přehrávač médií. Podporuje vytváření a produkci multimediálních projektů, dokonce se věnuje otázce přenosu vytvořených multimédií. Poskytuje podporu kompletního procesu zpracování obrazu: zachycení obrazu v reálném čase, programovou syntézu, import a export existujících multimediálních celků, úpravu, skládání a kompresi.

3.1.1 Sady nástrojů

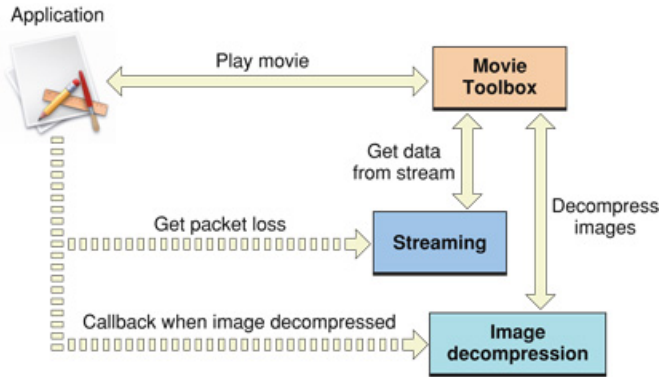
K podpoře kompletního spektra multimediálních úkolů, obsahuje QT sbírku nástrojů jako je Movie toolbox, Image Compression Manager, Sequence grabber a QuickTime streaming API. Názvy jednotlivých nástrojů většinou ponechám v anglické podobě, tím bude zaručena korespondence s originálními ilustracemi. Stejně jako v celé technické praxi i zde platí, že počeštění odborných názvů by mohlo více uškodit než pomoci.

Movie Toolbox slouží pro inicializaci, otevírání, přehrávání, editace, ukládání filmů a manipulaci s multimédii, které se mění s časem.

Image Compression Manager obsluhuje kompresi a dekompresi obrázků nezávisle na zařízení nebo na ovladačích.

Sequence grabber zachytává vzorky z videa nebo zvukové karty v reálném čase.

Streaming API umožňuje posílat a přijímat data přes datovou síť prostřednictvím protokolu RTP (Real-time Transport Protocol, přenos v reálném čase).



Obr. 3-4 Znázornění práce s nástroji QT, [9]

QT obsahuje další nástroje jako QuickTime VR (virtuální realita), sprite toolbox (obsluha obrazových objektů) a další, které však běžný uživatel znát nemusí, neboť se s nimi nesetká. Nástroje pracují dohromady tak, aby se uživatel mohl soustředit na práci, a nemusel se soustředit na chod celého QT. Různé nástroje často sdílí stejné datové typy a programové formy, tím je snazší vytvoření určité činnosti QT.

Množství nástrojů je užitečné pro přímý přístup k činnostem, které jsou jinak vykonávány automaticky. Například když použijete Movie Toolbox pro přehrání filmu, může to otevřít proud dat v reálném čase a dekomprimovat sérii obrázků bez potřeby, abychom obsluhovali Streaming API nebo Image Compression Manager. Pokud ale potřebujeme zkontrolovat ztracené pakety při dekompresi obrázku, můžete využít příslušný nástroj přímo z aplikace. Tento příklad je ilustrován právě na obrázku 3-4, kde je plnou čarou zobrazena automatická činnost QT a čárkovaně je znázorněna uživatelská obsluha jednotlivých nástrojů přehrávání.

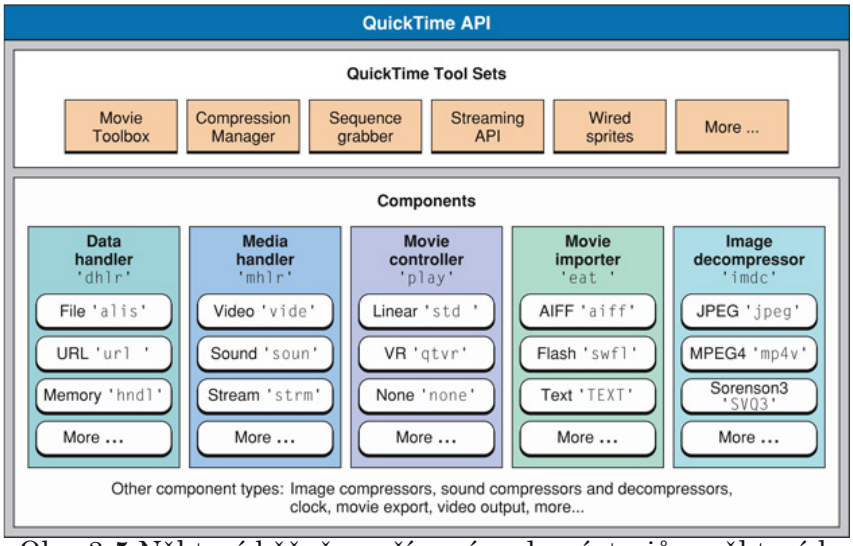
3.1.2 Komponenty

Architektura QT využívá široké množství komponent, které jej tvoří modulární, flexibilní a kdykoliv rozšiřitelný. QT komponenta je zdrojový kód pro obsluhu určité procedury a je sdílena jednotnou API. Je možné přidávat nové komponenty do QT tak, aby je existující aplikace mohly najít a použít je, když budou potřeba. Přidání komponent je možné za podmínky, odpovídají-li té samé API jako existující komponenty obecného typu.

Například, QT pracuje s množstvím multimediálních typů: zvuk, video, text, Flash, 3D modely, fotografická virtuální realita a další. Každý multimediální typ je podporován „media handler component" (tedy jakýmsi ukazatelem na daný typ média). Počet a typy podporovaných multimédií stále roste. Vytvořením media handler komponenty můžete vytvořit i nový typ pro QT.

Jsou zde další typy komponent pro ovládání a přehrávání filmů, importu a exportu multimédií, kompresi a dekompresi obrázků a zvuku, přístupu dat (ze souborového systému, síťových serverů nebo z pamětí), zachytávání sekvencí vzorkovaných digitálních dat a tak podobně.

Každá komponenta má typ, subtyp a „výrobní" kód, každý je reprezentovaný čtyř písmenným kódem. Čtyř písmenný kód je 32-bitová hodnota, která je obvykle nejlépe interpretována jako čtyři ASCII znaky. Například komponenta dekomprese obrázku má kód ‚imdc'.



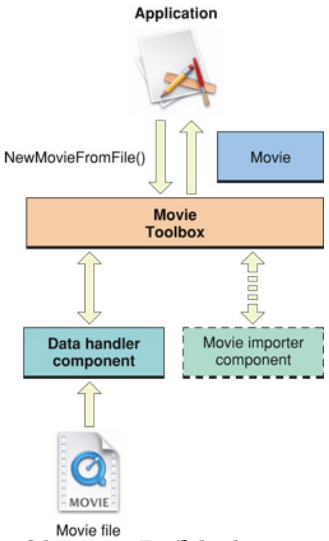
Obr. 3-5 Některé běžně používaná sady nástrojů a některé komponenty, [9]

QT má často vícenásobné komponenty daných typů. Například QT má mnoho komponent dekomprese. Všechny mají stejný typ: ‚imdc' . Každá ‚imdc' komponenta má svůj subtyp, který specifikuje způsob komprese. Například dekompresor obrázků JPEG má typ, subtyp a výrobní kód následující ‚imdc', ‚jpeg', ‚aapl'. Tříprvkové kódy jsou ve skutečnosti čtyř písmenné, kde čtvrté písmeno je ASCII znak pro prázdnou mezeru. Znak mezery je důležitou součástí a nelze ho vynechat.

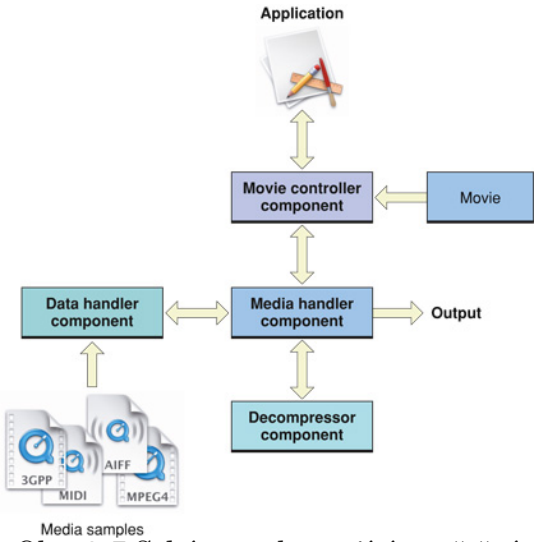
QT najde, vybere, načte a ukončí komponenty jak je potřeba. Děje se tak často skrytě na pozadí běhu aplikace. Například pokud QT otevře film, automaticky najde a načte správné ovladače médií a dekompresory filmu. Pokud se operace nezdaří, objeví se chybová hláška, jinak vše probíhá automaticky.

Aplikace řekne Movie Toolbox, aby načetl film (v tomto případu ze souboru). Data handler component vytvoří přístup k souboru, vybere se i správná komponenta k získání filmu ze souboru. QT vybere vyhovující komponentu na základě analýzy zdroje dat. Jestliže není soubor formátu MOV, movie importer component vytvoří ze vstupu filmový soubor podobný struktuře MOV. Odlišná komponenta bude použita pro import z MP3 souboru nebo JPEG, komponenty pro zpracování jsou vybrány na základě analýzy typu souboru, koncovky nebo MIME typu. Nakonec Movie Toolbox přenese film do aplikačního prostředí.

V aplikaci je filmový soubor načten a čeká se na stisk tlačítka „Play“ na ovládacím panelu (obsluhu zajišťuje movie controller). Media handler component se používá pro práci se všemi typy multimédií. QT vybere vyhovující media handler na základě analýzy načteného souboru.



Obr. 3-6 Příklad načtení filmového souboru, [9]



Obr. 3-7 Schéma zobrazující spuštění filmového klipu, [9]

Vzorky mohou pocházet z různých datových zdrojů, například film na lokálním disku může ukazovat na část souboru na lokálním disku, vzdáleném serveru i na Internetový stream. QT vybere odpovídající data handler pro každý datový zdroj. Media handler typicky volá decompressor component pro převedení multimediálního vzorku a vybere odpovídající dekompresor na základě typu multimédií a podle kompresního schématu. Media handler pak může poslat výstup dekompresoru přímo na nízkoúrovňové služby, jako je Sound Manager (zvukový ovladač), ke konečnému zpracování nebo poslat další komponentě, jako je video output component (video výstup), pro další úpravy.

3.1.3 Výstup

QT může poskytovat různé druhy výstupu během přehrávání. Výstupem je typicky zvuk a video s viditelnými ovládacími prvky. Může se ale jednat i o prostý zvuk bez viditelného přehrávání a ovládacích prvků, dokonce může jít o nevizuální a tichý DV (Digital Video) přenos přes FireWire port.

Aktuální výstup je ovládán nízkoúrovňovou technologií prostřednictvím Core Image, OpenGL, Core Audio, DirectX nebo Sound Manager. Uživatel ve většině případů tyto prostředky neovládá přímo. QT automaticky vybere a nakonfiguruje na dané platformě výchozí zařízení pro přehrávání. Avšak pokud je potřeba pracovat na nízké úrovni, například upravovat konkrétní video snímek během přehrávání nebo přidat filtr zvukového výstupu, není problém pracovat se základními, nejnižšími vrstvami QT. Je to závislé na použité platformě a revizi softwaru.

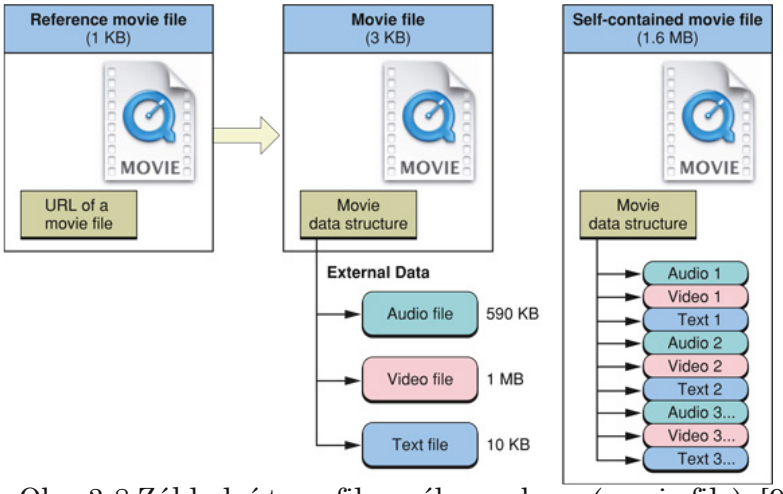
Například výchozí vizuální výstup QT 7 pro Mac OS pracuje v grafické souvislosti se systémem Quartz, zatímco vizuální výstup QT pro Windows a starší verze pro Mac OS pracuje na grafickém portu asociovaném s okny reprezentovaným pamětí (bufferem) zvanou GWorld. Doplňkem v QT 7 můžeme vytvořit vizuální kontext, kterým lze specifikovat částečný formát výstupu, příkladem je buffer pixelů pro Core Image nebo OpenGL textura. Tento postup znamená, že uživatel sám vykreslí dekomprimované snímky na obrazovku. K vykreslení slouží některá základní grafická vrstva Mac OS API jako je Core Video a Core Image, OpenGL nebo QuickDraw, ve Windows se používá přímo Windows video API, OpenGL nebo části QuickDraw.

3.1.4 QuickTime movie

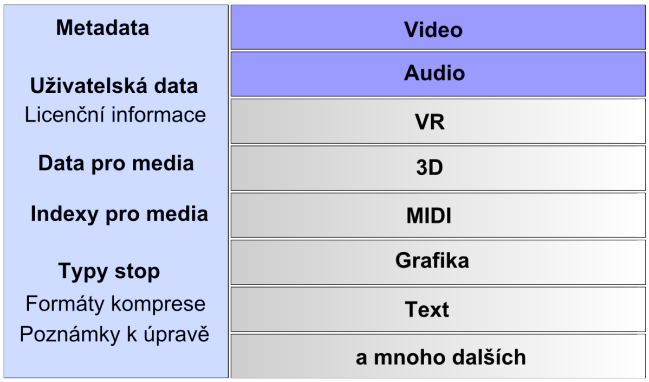
Nejedná se o film jak by se podle překladu mohlo zdát, tímto termínem je označována datová struktura, se kterou pracuje QT. V QT movie najdeme popis, jak přistupovat k multimediální prezentaci uložených dat, tzn. o jaký typ média se jedná, kde jsou uloženy datové části souboru a jak poskládat jednotlivé komponenty do výsledného celku. Neobsahuje tedy přímo video nebo audio vzorky, ale poukazuje na umístění dat a jak s daty dále pracovat.

Spojení „QuickTime movie“ nalezneme především v dokumentaci firmy Apple, běžně se totiž pod anglickým slovem „movie“ označuje filmový soubor. V rámci rozlišení datové struktury a zdrojového souboru s filmem jsme nuceni použít pro soubor s video daty méně obvyklé (delší) spojení QuickTime movie file.

Filmový soubor může mít tři různé podoby a to podle toho, v jaké podobě je uložena informace o multimediálních datech. V prostředí Internetu nejčastěji narazíme na první typ obsahující pouze referenci, druhý typ obsahuje datovou strukturu, nikoliv však přímo stream a poslední typ je nejčastěji používaný soubor obsahující všechna data a streamy v jediném souboru.



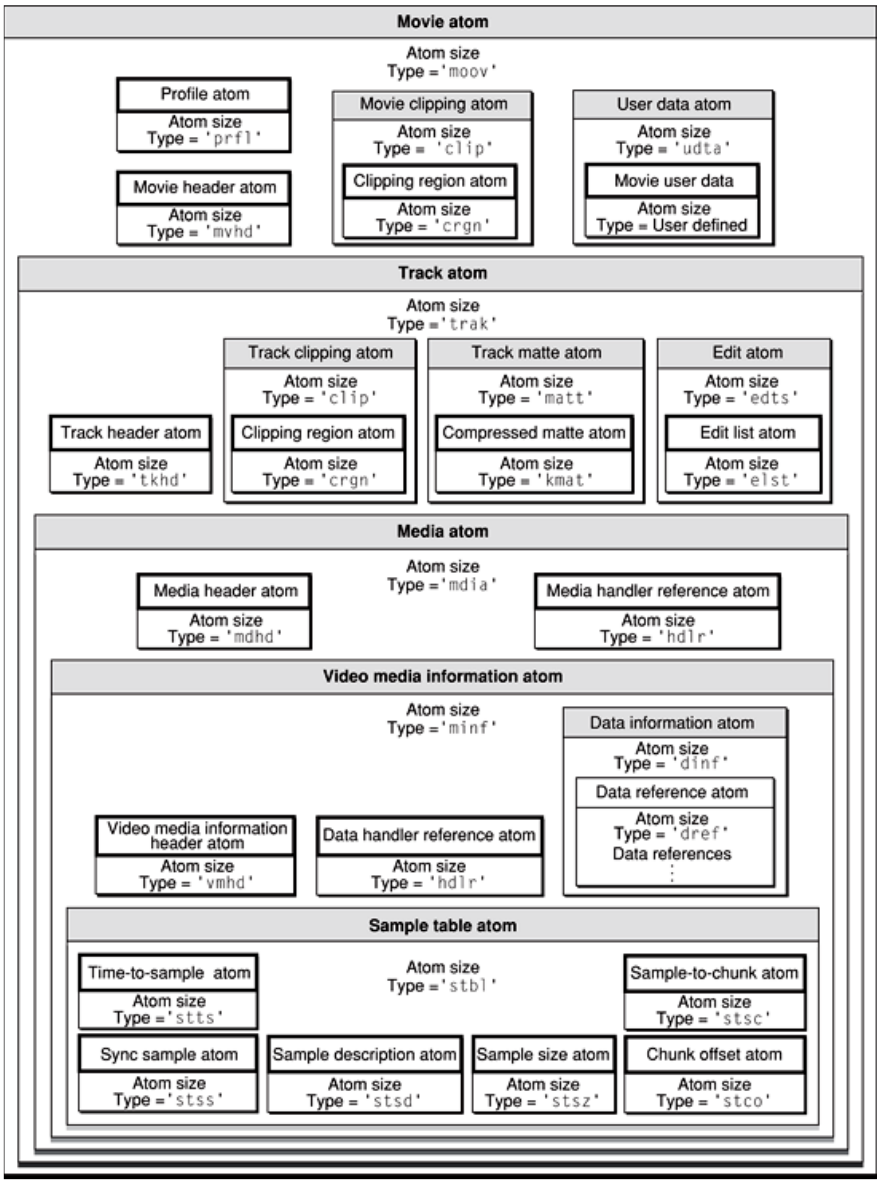
Obr. 3-8 Základní typy filmového souboru (movie file), [9]



Obr. 3-9 Jednoduchá struktura souboru formátu MOV (vlevo) a způsob řazení jednotlivých typů streamů (vpravo), [10]

3.1.5 Struktura souboru MOV

Základním prvkem souboru QuickTime je atom. Atom je jednoduchá struktura obsahující základní popis (typ a velikost dat, která obsahuje) a samotná data. Z atomů lze vytvořit hierarchickou strukturu, protože jeden atom může obsahovat jiný. Tímto postupem můžeme vytvořit poměrně složitou strukturu nesoucí dostatečné množství přehledně uspořádaných informací. Nejvyšším atomem a tedy i základní strukturou souboru QuickTime je atom označovaný typem ‚moov'.



Obr. 3-10 Detailní struktura souboru – atom ‚moov‘, [9]

3.2 FFmpeg

FFmpeg je sbírka volně dostupných knihoven, které umí zaznamenávat, konvertovat a streamovat digitální audio a video. Nejdůležitější knihovna audio/video kodeků je libavcodec. FFmpeg je vyvíjen pod Linuxem, ale může být sestaven pod většinou operačních systémů, včetně Windows. Stojí za zmínku, že většina vývojářů FFmpeg, je současně vývojovým týmem projektu MPlayer. Proto je také FFmpeg hostován na serverech MPlayeru.

Framework tvoří několik komponent:

ffmpeg – pomůcka příkazového řádku pro přeměnu jednoho video formátu na jiný. Je také podporováno stahování a kódování z TV karet v reálném čase

ffserver – HTTP(případně RTSP, Real-time Streaming Protocol) multimediální server streamingu pro živé vysílání. Je podporováno časové posouvání živého vysílání.

ffplay – jednoduchý multimediální přehrávač založený na SDL (Simple DirectMedia Layer) a na knihovnách FFmpeg

libavcodec – knihovna obsahující všechny audio/video kodéry a dekodéry. Většina kodeků je vyvíjena od začátku, aby byl zajištěn nejlepší výkon a vysoká míra přístupnosti.

libavformat – knihovna obsahující demultiplexory a multiplexory pro audio/video záznamové formáty

libavutil – pomocná knihovna obsahující programové postupy společné různým částem FFmpegu

libpostproc – knihovny obsahující programové postupy video postprocesingu

libswscale – knihovny obsahující programové postupy škálování video obrazu

FFmpeg je uvolněn pod GNU LGPL nebo GNU GPL (podle podknihoven, na kterých závisí). Nejsou zde žádná formální vydání. Na místo toho doporučují vývojáři používat vždy poslední verzi (respektive subverzi). Volná dostupnost knihoven FFmpeg je jednou z hlavních výhod tohoto frameworku. Další výhodou je možnost úpravy všech knihoven včetně *libavcodec*. Popularita je zajištěna především bezchybnou prací s kodeky Xvid. FFmpeg je dokonce doporučován v souvislosti s šířením komprimovaných video souborů, neboť v sobě zahrnuje vedle kodeků Xvid a DivX mnoho dalších filtrů pro přehrávání multimediálních souborů.

3.3 Helix DNA

Helix je otevřený více formátový multimediální framework. Kód je uvolněn v binární podobě formou zdrojových kódů pod různými licencemi. Především je používán firmou RealNetworks. Helix DNA klient a Helix Player jsou licencovány pod GNU GPL. RealNetworks přispěl svými zdrojovými kódy i do komunity Helix, aby podpořil rozvoj průmyslu směrem k multiformátům pro různé platformy a využití víceprocesorových systémů.

Helix DNA klient je prostředek pro přehrávání a činnosti digitálních médií.

Přehrávač Helix je open source multimediální přehrávač postavený na klientu Helix DNA pro Linux. Přehrávač Helix slouží jako základ pro RealPlayer verze 10 a novější. Přehrávače pro Windows a Macintosh, stejně jako začleněná verze RealPlayeru pro mobilní telefony, jsou založeny na principiálně stejném přehrávači Helix. Zahrnuje podporu následujících médií:

audio formáty – Ogg Vorbis, .au

video formáty – Ogg Tudora, AVI

popisné formáty – SMIL, SDP

obrázkové formáty – JPEG, GIF, PNG

protokoly – RTSP, RTP, HTTP, Multicast, RDT (datový přenos Real)

Helix je jeden z méně rozvinutých frameworků. Jednak kvůli výhradnímu použití na systémech Linux a pak kvůli specifické podpoře kompresních formátů. Hlavní uplatnění nachází v přehrávačích RealPlayer. Streamová architektura předurčuje formát k přehrávání především po Internetu.

3.4 GStreamer

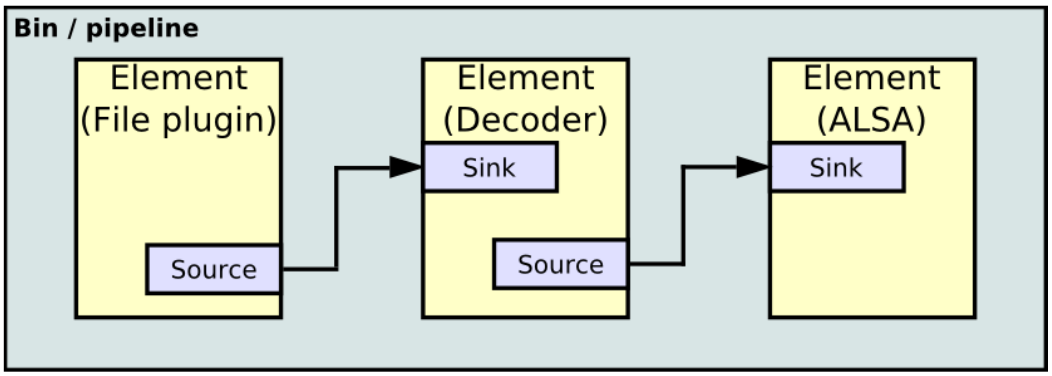
GStreamer je multimediální framework napsaný v programovacím jazyce C. GStreamer slouží jako hostitel multimediálním aplikacím, jako jsou video editory a přehrávače. GStreamer je nezávislí na platformě, funguje na Linuxu (x86, PowePC, ARM), Solarisu (x86, SPARC), Mac OS X, Windows a OS/400. Další výhodnou vlastností je možnost volného šíření, s licencí pod GNU LGPL.

GStremer je implementován v linuxovém pracovním prostředí GNOME. GNOME zahrnuje GStreamer od verze 2.2. Právě díky linuxovým distribucím a podpoře aplikací GNOME a GTK+ je GStreamer hojně využíván a začíná se rozšiřovat i do jiných softwarových skupin.

Upravený GStreamer se také používá jako vestavěný software Maemo od firmy Nokia, který byl představen s mobilním telefonem Nokia 770.

Zásobník nebo linka se skládá z elementů – zásuvných modulů. Data proudí skrz linku v jednom směru. Elementy mají schopnost zvanou „zastřešení“.

Diagram může být příkladem přehrávání MP3 souborů použitím GStreameru. Zdroj souboru načte MP3 soubor z počítačového disku a pošle do MP3 dekóderu. Dekodér dekóduje soubor s daty a konvertuje je do PCM vzorků, které pošle do zvukového ovladače ALSA. Zvukový ovladač pošle PCM vzorky do reproduktorů.



Obr. 3-11 Diagram zpracování dat pomocí GStreamer, [11]

GStreamer používá architekturu zásuvných modulů, které tvoří většinu funkcionality. Moduly jsou implementovány ve sdílených knihovnách. Knihovny zásuvných modulů jsou dynamicky načítány, aby podporovaly široké spektrum kodeků, digitálních převodů a vstupně/výstupních ovladačů.

Úpravy knihoven jsou možné pomocí programovacích jazyků Python, C++, Perl, GNU Guile a Ruby.

3.5 DirectShow

Významným multimediálním frameworkem na platformě Windows je DirectShow. Jeho význam je důležitý zejména proto, že je obsažen implicitně v přehrávači Windows Media Player.

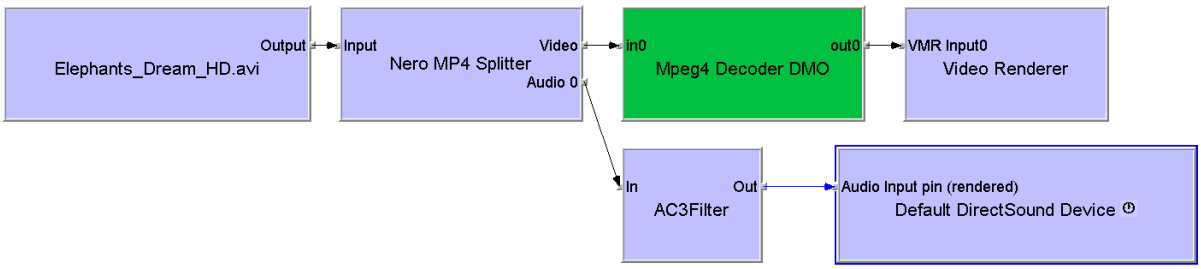
DirectShow (někdy zkráceně jako DS nebo DShow), kódovým jménem Quartz, je multimediální framework a API vytvořené firmou Microsoft pro softwarové vývojáře, aby sloužil k různým operacím s multimediálními soubory. Je náhradou pro dřívější technologii VfW (Video for Windows). DShow je založen na programovacím frameworku Component Object Model (COM) Microsoft Windows, díky tomu nabízí společné rozhraní pro média v mnoha programových jazycích Microsoftu. Vývojové nástroje DirectShow a dokumentace jsou poskytovány v softwarovém balíčku Microsoft Platform SDK (Software Development Kit) nebo přímo v balíčku DirectX SDK.

Návrhový model

DirectShow pracuje podle takzvaných schémat filtrů. Nejdříve analyzuje přehrávaný soubor a načte všechny potřebné filtry, které se v multimediálním souboru vyskytují. Poté spojí filtry dohromady a daný soubor přehraje.

Diagram filtrů lze i uživatelsky upravit, záleží pak na vývojáři, který filtr pro určitý formát souboru použije.

Přestože je DirectShow, součástí DirectX a tedy součástí platformy Windows, nemá příliš silnou pozici na poli multimediálních frameworků. Svědčí o tom i fakt, že se Microsoft chystá nahradit DirectShow formátem Media Foundation (od verze operačním systému Windows Vista).



Obr. 3-12 Graf filtru testovacího souboru MOV, vykresleno programem GraphEdit

3.6 KMixer

KMixer je ovladač Kernel Audio Mixer, část WDM (Windows Driver Model) Audio v různých verzích Windows, která obsluhuje směšování vícenásobného zvukového zásobníku na audio výstup.

Úkoly vykonávané KMixer.sys

- směšování vícenásobných PCM audio stop
- formátování, bitovou a vzorkovou konverzi
- nastavení reproduktorů a mapování kanálů

KMixer byl navržen, aby pomohl aplikacím ulehčit od směšování audio stop, zvláště na low-endových zvukových kartách, které nepodporují vícenásobné zvukové škálování. Nicméně to představuje významné problémy.

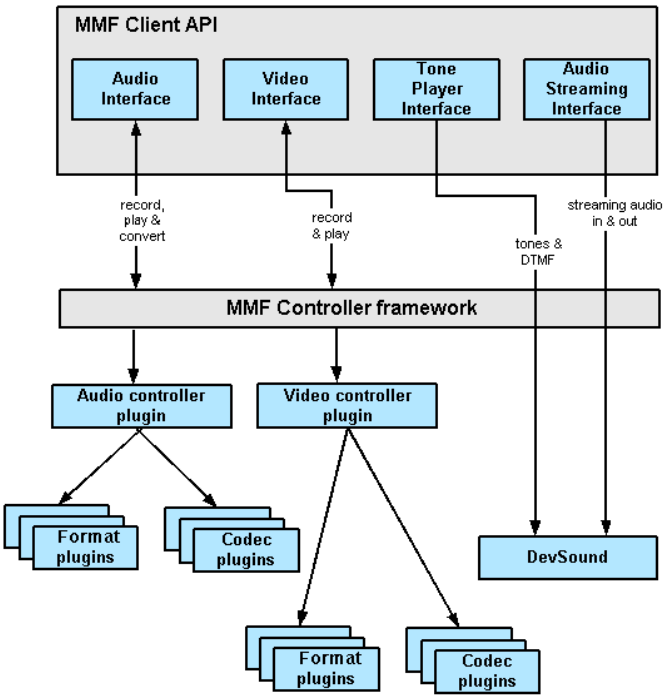
Zaprvé, zpoždění KMixeru je okolo 30ms a nemůže být zmenšeno, protože tato komponenta je umístěna právě nad porty zvukového ovladače, takže každá zvuková stopa, včetně těch vyslaných od DirectSound a WinMM, prochází přes jádro směšovače. Poté zkouší KMixer sloučit všechny datové formáty, které jím prochází, dokonce i ty, které nepodporuje. To způsobuje různé problémy s přehráváním filmů, kdy se KMixer snaží zprůchodnit AC3 zakódovaný prostorový zvuk přes S/PDIF výstup zvukové karty na externí vstup receiveru domácího kina. Řešením je nový jádrový mód API, Direct Kernel Streaming, který musí být zaveden, aby obešel KMixer a vyhnul se problémům směšování.

3.7 Multi Media Framework (MMF)

Multimediální framework pro přenosná zařízení. Je součástí operačního systému Symbian, kde poskytuje API (programovatelné rozhraní) pro práci se zvukem a obrazem. Toto rozhraní se označuje MMF Client API.

Vlastnosti MMF jsou rozděleny do 4 podřazených rozhraní, které obsahují konkrétní obslužné knihovny pro práci s audio a video soubory. Dále jsou využívány zásuvné moduly, ke kterým se přistupuje přes MMF Controller. Na obrázku je zobrazen také přímí přístup na nízkoúrovňové zvukové zařízení DevSound.

MMF je jednoduchý framework, se kterým se běžně setkáváme na moderních přenosných zařízeních jako jsou mobilní telefony a kapesní počítače PDA.



Obr.3-13 Struktura Multi Media Framework, [12]

4. Multimediální kontejner

Z českého slova kontejner snadno poznáme základní vlastnost, kterou je schopnost uložit různé „věci“ do jednoho přepravního boxu. S pojmem kontejner se setkáme v souvislosti s konkrétními typy souborů, tzn. že koncovka souboru poukazuje na jistý druh kontejneru. K objasnění přispěje výčet několika multimediálních kontejnerů.

4.1 AVI (Audio Video Interleave)

Jedná se o nejrozšířenější kontejner pro ukládání streamů, volně by se dal název přeložit jako „prokládané video a audio“. Podporuje většinu kompresních video formátů, s omezenými možnostmi uloží komprimované audio a umí zpracovat i titulky.

Na začátku souboru je hlavička, která obsahuje informace o videu (snímkovou frekvenci, rozlišení, kodek, bitrate atd.) a informace o zvuku (vzorkovací frekvence, bitová hloubka, kvantizace, kodek). Na konci souboru je tabulka, ve které jsou informace o pořadovém čísle jednotlivých snímků videa, popřípadě index audio paketu a jejich pozici v souboru.

Tento způsob rozvržení souboru má dvě velké nevýhody. Indexace na konci souboru neumožňuje přehrát soubor dříve, než je k dispozici celý, tedy kontejner není použitelný pro streamování. Druhá nevýhoda indexace je v tom, že čas snímku není určen absolutně, ale lze ho spočítat pouze ze snímkové frekvence videa a jeho indexu. Problém nastává při rozdílné časové základně, kdy je audio dat více (méně) než video dat. Příkladem je zachytávání videa na TV kartu a zvuku na zvukovou kartu, ačkoli vypadá časová základna stejně, dochází k drobným nepřesnostem v synchronizaci audio – video. Hlavní výhodou AVI je velká kompatibilita s operačními systémy a velká podpora pro přehrávání.

```
RIFF ('AVI'
LIST ('hdrl'
'avih'()
LIST ('strl'
'strh'()
'strf'()
'strd'()
'strn'()
...
)
...
)
LIST ('movi'
{SubChunk | LIST ('rec'
SubChunk1
SubChunk2
...
)
...
}
...
)
['idx1']
)
```

Obr. 4-1 Struktura AVI, [13]

Existují tři verze kontejneru AVI :

AVI 1.0 umožňuje nahrávat pouze do velikosti 1GB, maximální počet snímků je 22500, tzn. asi čtvrt hodiny záznamu pro 25fps, používal se ve Windows 3.1,

AVI 1.1 rozšířeno nahrávání a indexování do velikosti souborů 2GB, toto omezení se překonává rozdělením na více souborů

AVI 2.0 označuje se také OpenDML, má neomezenou velikost souboru

AVI soubor je založen na formátu RIFF (Resource Interchange File Format), začíná vždy hlavičkou identifikující soubor, hlavička začíná písmeny RIFF, velikostí souboru a typem, dále jsou informace o typu obsažených streamů a multiplexovaná data. Na konci je zmíněný index, bez kterého není přehrání možné. Video a audio data jsou multiplexována po snímcích a audio paketech, normálně je to pravidelně například po snímku nebo po určitém čase (doporučená hodnota 500ms), ale obecně to může být jakkoliv. S tím souvisí nastavení prokládání při zachytávání videa – nastavení „Full“ prokládá po nastaveném čase, „None“ prokládá okamžitě po přijetí dat, tedy není zaručeno pravidelné prokládání, „Capture“ prokládá po nastaveném čase, ale pokud by to narušilo tok dat, tak nemusí prokládání dodržet.

4.2 MPEG

Kontejner obsahující video s kompresí MPEG-1, MPEG-2 a audio ve formátu AC3 nebo MP2. Formát MPEG byl rozšířen především v éře VideoCD a DVD.

Standard, který opět definuje multiplexované zachytávání audio vizuálních streamů. Umožňuje kromě off-line uložení i streamování přes internet, satelitní vysílání a může pojmout i interaktivní obsah. Formát MPEG tvoří několik aplikačních vrstev.

Elementary stream (ES) je nejnižší aplikační vrstva. Definuje vytvoření samostatného streamu z jednoho zdroje (video, audio a jiné). Dle obsahu se rozděluje několik forem ES - Digital Control Data, Digital Audio, Digital Video, Digital Data (synchronní nebo asynchronní). Data videa a zvuku jsou organizovány do částí access units (AE), u videa tvoří AE jeden obrazový snímek.

Packetised Elementary Stream (PES) je navazující vrstva, do které je převedena posloupnost paketů tvořených z několika AE. Maximální velikost PES paketu je 65536 bytů a obsahuje hlavičku o streamu a data (Elementary Stream). Pokud uložíme samotný PES do souboru, používají se přípony vyjadřující daný obsah - MPV (MPEG video), M2V (MPEG-2 video), MPA (MPEG Audio), MP2 (MPEG Audio Layer 2), MP3 (MPEG Audio Layer 3).

MPEG Program Stream (PS) je metoda multiplexování několika streamů z vrstvy PES do jednoho výsledného souboru. Čas paketů řazených za sebe je synchronizován k jedné časové základně. Tento typ se používá v prostředí se zaručenou bezchybností (záznamová média DVD a při přenosu po Internetu v souborech MPG a M2P).

MPEG Transport Stream (TS) je druhá metoda multiplexování streamů, u kterých však nelze zaručit bezchybnost (např. záznam z DVB - Digitální televizní vysílání). Pakety PES se rozdělí na fixní bloky o velikosti 188 bytů a zabezpečené se přenášejí po síti. Přenos má konstantní datový tok a umožňuje multiplexovat i více televizních programů v jednom kanálu.

4.3 ASF (Advanced Systems Format)

Formát z dílny společnosti Microsoft. Původně byl navržen pro odstranění nedostatků formátu AVI a určen jako formát pro streamování přes Internet. Hlavní nevýhoda tohoto formátu je v podpoře omezeného počtu kompresních formátů. ASF je koncepčně navržen tak, aby pracoval výhradně jen s kodeky firmy Microsoft (Windows Media Video a Audio). Dále je licenční politikou zamezena práce se soubory přímo, použít se musí DirectShow filtry. Struktura je založena na objektech a tyto základní složky jsou číselně identifikovány. Nejvyšší vrstva obsahuje tři objekty. V Header Object je uložena hlavička a podle identifikátoru poznáme typ souboru.

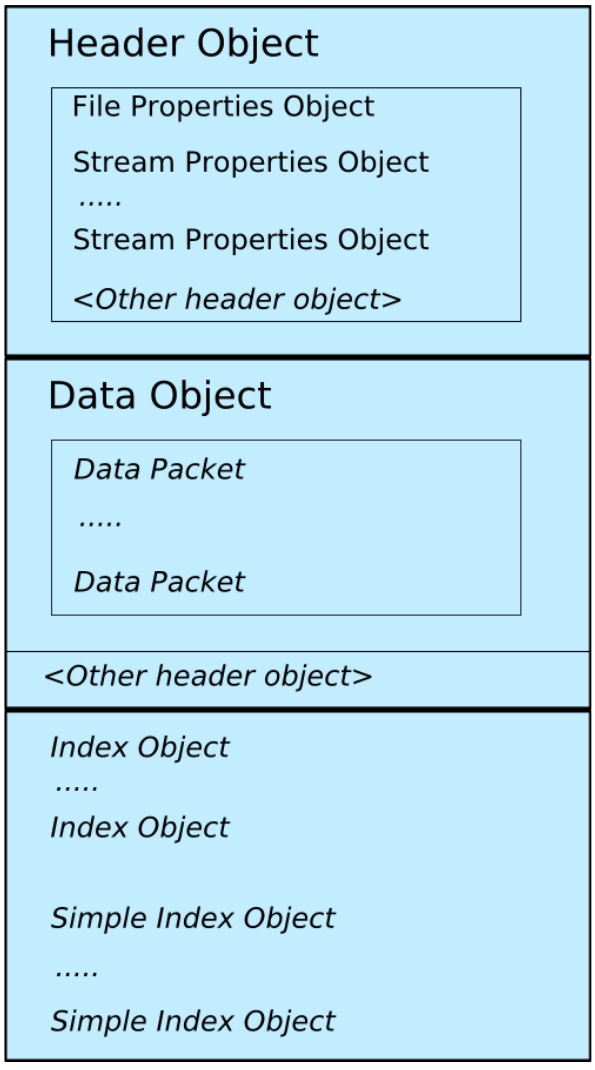
File Properties Object - základní statistika o souboru jako délka, velikost, datum vytvoření, jedinečné ID souboru atd.

Stream Properties Object - je zde povinně pro každý stream v souboru a obsahuje informace o typu streamu a formátu

Header Extension Object - je zde pro další možné rozšíření specifikace

Dále hlavička obsahuje upřesňující nepovinné vlastnosti souboru:

Codec List Object - zobrazitelné informace o kodecích, které jsou použity pro přehrání souboru



Obr. 4-2 Struktura ASF souboru, [13]

Script Command Object - seznam příkazů synchronizovaných na určitý čas, obsahuje buď URL adresu nebo soubor

Marker Object - jde o časové značky pro rychlý přesun v souboru, obdoba kapitol u DVD

Bitrate Mutual Exclusion Object - identifikuje streamy, které jsou vůči sobě vylučující, např. video stopa v určitém čase nemůže být přehrávaná zároveň s jinou, umožňuje uživateli vybrat například video s různým bitrate apod.

Error Correction Object - identifikuje metodu korekce chyb

Content Description Object - popis obsahu jako autor, titul, copyright, apod., obdoba ID3 tagu u MP3

Extended Content Description Object - rozšíření Content Description o další informace

Stream Bitrate Properties Object - informace o průměrném bitrate každého streamu

Content Branding Object - je možné připojit značku/banner, který se

bude zobrazovat při použití souboru, lze i pomocí URL

Content Encryption Object - informace o použitém DRM

Extended Content Encryption Object - možnost chránit soubor pomocí Windows Media Rights Manager 7

Digital Signature Object - podpis, který chrání informace v hlavičce

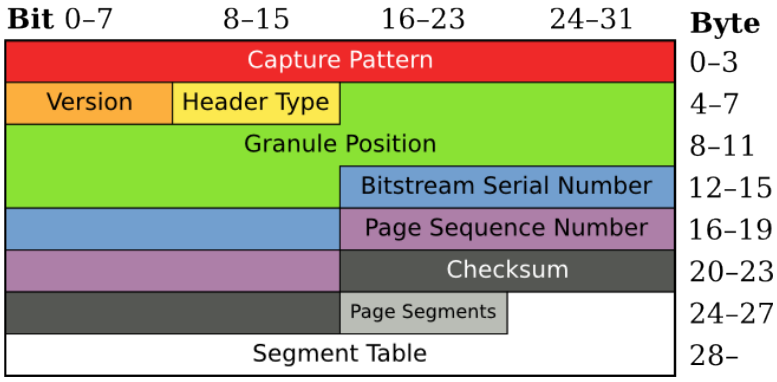
Padding Object - umožňuje zvětšit velikost hlavičky o libovolnou velikost, nenese žádná užitečná data

Data Object je další vrstva obsahující samotné multiplexované streamy v podobě Data Packets. Pakety mají vždy stejnou velikost a obsahují informace pro jeden stream (někdy i pro více streamů). Setříděny jsou podle času kdy začínají a končí, opatřeny jsou buď indexem podle fixního času Simple Index Object, nebo mají navíc index i podle čísla snímku a označují se jako Index Object.

4.4 OGG

Formát nadace Xiph.org, který je součástí myšlenky o vytvoření svobodného softwaru. OGG je definován v dokumentu RFC 3533. Název vznikl podle postavy z knížního souboru s názvem Zeměplocha od britského autora Terryho Prachetta.

Jednotlivé části dat se nazývají stránky a každá začíná identifikačním řetězcem „OggS“. Dále každá stránka nese informace o svém sériovém číslu a číslu stránky, aby ji bylo možné identifikovat v sérii stránek, které dohromady tvoří datový proud. Množství datových proudů je pak multiplexováno do souboru. Datový proud může být i zřetězen k jinému již existujícímu souboru. Pro kódování a dekódování dat slouží knihovna libogg.



Obr. 4-3 Struktura hlavičky souboru OGG, [8]

Capture Pattern - 32 bitů
Vzor zachytávání nebo synchronizační kód je číslo zajišťující synchronizaci při rozkladu Ogg soubor. Pomáhá opětovné synchronizaci v příadě, že byla data poškozena, kontroluje se tak soubor před zahájením rozboru stránkové struktury.

Version - 8 bitů; Určuje verzi použitého datového proudu OGG formátu. Zatím se zapisuje číslo 0, jedná se o položku s ohledem na budoucí vývoj.

Header Type - 8 bitů; Příznak ukazující na typ stránky, který následuje. Hodnota 0x01 značí pokračování v datovém proudu, hodnota 0x02 neboli BOS (Beginning of Stream) značí první stránku logického datového proudu a naproti tomu s hodnotou 0x04 je příznak EOS (End of Stream) značící

konec stránky.

Granule Position - 64 bitů; Je to časová značka v Ogg souboru. Udává abstraktní hodnotu, jejíž význam je určen kodekem. To může být například počet snímků, počet obrazových rámců nebo se může jednat i o komplexnější schéma.

Bitstream Serial Number - 32 bitů; Tímto číslem se identifikuje stránka patřící jednotlivým datovým proudům. Toto pole umožňuje, aby byla stránka doručena příslušnému dekóderu. Podle toho se například rozhodne, že bude jeden proud audio (vorbis), a další bude video (theora).

Page Sequence Number - 32 bitů; Celé číslo, které se automaticky zvyšuje a udává pořadí jednotlivých datových proudů. Lze tak dohledat, pokud se nějaký proud ztratí.

Checksum - 32 bitů; Pole s kontrolním součtem. Pokud jsou data změněna, přepíše se nulová hodnota a daná stránka se zahodí.

Page Segments - 8 bitů; Udává počet částí dané stránky a také kolik bytů následuje v poli Segment Table. Počet částí jedné stránky může být maximálně 255.

Segment Table – Udává délku jednotlivých částí (segmentů).

Segmenty se mohou seskupovat do paketů. Pokud má segment délku 255 dává najevo, že je částí nějakého paketu. Délka 1 – 254 určuje samostatný segment. Konečný segment má délku nula.

4.5 MP4

Multimediální kontejner definovaný standardem ISO. Je také známý pod označením MPEG-4 Part 14. Základ tvoří kontejner MOV používaný v multimediálním frameworku QuickTime. Jako alternativa k zastaralému kontejneru AVI má několik výhod. Může pojmout video, menu, titulky, více zvukových stop i 3D objekty. Umožňuje také streamování videa.

Použitelné kompresní formáty pro obraz jsou MPEG-1, MPEG-2, MPEG-4 a pro zvuk MP3, ACC. MP4 je primárně určen pouze pro kompresní formáty skupiny MPEG. Existují i rozšíření, a to oficiální i neoficiální. Mezi neoficiální modifikace patří i úpravy hlavních propagátorů tohoto formátu – společnosti Nero a Apple. Nero do svého rozšíření implementuje například titulky ve formátu VobSub, Apple začlenil do multimediálních souborů ochranu DRM. Oficiální modifikace je známá jako 3GPP, což je formát vytvořený při zavádění mobilní sítě třetí generace. Standardizuje pro obraz kompresi H.263 a pro zvuk AMR (Adaptive Multi-Rate), což je kodek pro kompresi řeči.

Kontejner poznáme podle koncovky .mp4 nebo .3gp. Přestože v ISO standartu najdeme jen příponu mp4, nalezneme přípon více.

m4a – obecný zvuk bez ochrany

m4p – zvuk s ochranou DRM (nalezneme jej např. na iTunes)

m4b – rozšířený zvukový soubor, dokáže využít vlastností značek (bookmark)

m4r – vyzváněcí melodie na mobilní telefony iPhone

3g2 – pro data v mobilních zařízeních, která obsahují jiný formát než MPEG

4.6 Matroska

Vývojová skupina CoreCodecs zřejmě ne náhodou použila název pro multimediální kontejner Matroska. Nápad vzešel z ruské hračky známé i v našich končinách. Matrijoška je dřevěná malovaná panenka skládající se z několika částí, které jsou podle velikosti v sobě ukryty a tvoří navenek jeden celek. Projekt Matroska byl založen koncem roku 2002.

Obr. 4-4 Matrijoška, inspirující hračka

Multimediální kontejner, který pojme většinu dnešních kodeků (ať už video či audio). Matroska je založená na kódu EBML (Extensible Binary Meta Language). Jedná se o otevřený standardní projekt pod licencí GNU LGPL. Jazyk EBML dává souborům Matroska derivované vlastnosti formátu XML. Uspořádání je bitové nebo oktanové. EBML obecně popisuje techniku bitového značkování. Tento způsob je, stejně jako u XML, netečný k datům, které obsahuje. V tomto je Matroska specifická – snaží se definovat podmnožinu EBML vzhledem k audio a video datům.

V dnešní době je formát MKV nejčastěji spojován s přenosem formátu HD. Tímto způsobem se dostal kontejner Matroska do podvědomí uživatelů a má pravděpodobně větší šance na existenci než kontejner AVI.

Obr. 4-5 Jednoduchá reprezentace souboru Matroska, [14]

Header – zahrnuje verzi EBML, v které je soubor vytvořen, a jaký je to EBML typ (v tomto případě soubor typ Matroska)

Metaseek - seznam, kde se nacházejí ostatní části souboru. Technicky není potřeba, ale pro práci s daty je užitečný. Pořadí jednotlivých částí není striktně dáno. Např. kapitoly se mohou nacházet uprostřed datového rámce

Segment Information - základní informace o souboru – název, ID (v případě řady souborů i ID následujícího). Pomocí ID se stává soubor světově jednoznačný.

Track - základní informace o stopách – rozlišení videa, vzorkovací informace o audio stopě apod. Také má zapsaný kodek potřebný pro přehrání zaznamenané stopy.

Chapters – seznam kapitol

Clusters - všechna data – video snímky a zvukové stopy, pozice dat je časová, tedy nedochází k problémům se synchronizací

Cueing Data – část obsahující cue – indexy všech stop. Je to podobné jako Metaseek, ale slouží pro vyhledávání během přehrávání. Bez této možnosti by vyhledávání bylo obtížné a musel by se celý soubor pokaždé prohledávat až na určený časový kód.

Attachment – sekce pro připojení jakéhokoliv souboru k soboru Matroska. Lze připojit různé soubory od obrázků, přes webové stránky až po kodeky potřebné pro přehrávání.

Tagging – obsahuje všechny tagy náležící všem stopám souboru

5. Praktická část

Na třech příkladech ukážeme praktické použití multimediálního frameworku. Pro demonstraci je vybrán jeden framework ze skupiny, dle rozdělení v úvodu 4. kapitoly. Jedná se o QuickTime, DirectShow a GStreamer. V prvních dvou případech se jedná o aplikace s grafickým rozhraním, třetí zmíněný je konzolová aplikace.

5.1 Jednoduchá aplikace QuickTime

K naprogramování aplikace jsou použity dokumenty [19], [20], [21] a ukázkové zdrojové kódy [22]. Aplikace slouží k seznámení s multimediálním frameworkem QuickTime po vývojové stránce. Protože aplikace nenese písemný souhlas společnosti Apple, je nakládání s dokumenty k vytvoření aplikace QuickTime [19] i obsahem proti licenčnímu ujednání. Licence se nevztahuje na použití informací pro osobní potřebu jednotlivce na jediném počítači a to včetně použití ukázkových kódů, tak je možné otestovat si framework alespoň soukromně mimo komerční sektor.

Základní software potřebný k programování QuickTime je vývojový balíček QuickTime SDK (Software Development Kit) pro Windows. Balíček obsahuje základní dokumentaci, knihovny (formát LIB), hlavičkové soubory a knihovny rozhraní API. Zdrojové soubory jsou napsány v jazyce C. Při programování v C (popř. v C++) je hlavním nástrojem při tvorbě multimediálního přehrávače QuickTime třída QTML (QuickTime Media Layer), konkrétně QTMLClient.lib a hlavičkový soubor QTML.h.

Při programování aplikace není potřeba myslet na ovládací prvky pro přehrávání, neboť QT obsahuje interface včetně jednoduchého ovládání. Pro ověření základních vlastností postačí vytvoření jednoduchého menu, které obsahuje položky pro otevření souboru, uložení a ukončení aplikace. Tlačítko pro spuštění/zastavení videa a ovládání hlasitosti bude vytvořeno spolu s oknem videa.

Nevýhoda programování pod Windows je převod mezi procedurami MAC OS a Windows. Koncept Windows používá například Message, kdežto v systému Apple pracujeme s Event, Window Handle se změní ve Window Pointer a další. Struktura fungujícího přehrávače musí obsahovat tyto body (programování pomocí QTML):

1)Na začátku inicializaci QTML a QuickTime movie (viz. 3.1.4) (*InitializeQTML, EnterMovies*)

2)Asociace grafického portu QuickDraw s oknem filmu (*CreatePortAssociation*)

3)Otevření souboru s filmem a extrahovat ho do přehrávače v podobě QuicTime movie (*OpenMovieFile, NewMovieFromFile*)

4)Vytvoření ovládače pro QuickTime movie (*NewMovieController*)

5)V procedurách oken přeměnit příchozí zprávy na události QTML a ty použít při zpracování přehrávačem (*WinEventToMacEvent, MCIsPlayerEvent*)

6)Uvolnění QuickTime movie a jeho ovladače, když už nebudou potřeba (*DisposeMovie, DisposeMovieController*)

7)Zavření grafického portu pro QuickTime movie po ukončení okna (*Destroy-PortAssociation*)

8)Odstranit QuickTime movie a ukončit QTML (*ExitMovies, TerminateQTML*)

Podle návodu k tvorbě multimediálního přehrávače QuickTime pro operační systém Windows [19] lze naprogramovat funkční jednoduchý přehrávač pracující se soubory ve formátu MOV. Podporovány jsou téměř všechny dostupné kompresní formáty jako u běžného přehrávače QuickTime.

Praktická ukázka přehrávač video souborů se základními funkcemi. Naprogramován je v prostředí Microsoft Visual C# 2008 Express Edition. Jednoduché okno aplikace obsahuje roletové menu, které umožňuje základní manipulaci s video soubory. Ovládací prvky videa jsou součástí komponenty QuickTime. Komponentu QuickTime musíme nejprve zpřístupnit. To provedeme ve správci všech komponent prostým zaškrtnutím Apple QuickTime Control. Komponenta je poté k dispozici v nabídce nástrojů (Toolbox). Po vložení prvku QuickTime do formuláře se vytvoří reference na tři knihovny dll – AxQOControlLib, QTOControlLib a QTOLibrary. Tyto knihovny nám umožňují využít všech vlastností frameworku QuickTime.

Knihovna AxQTOControlLib je pomocná knihovna a dovoluje nám zacházet s frameworkem QT jako kdyby se jednalo o standardní prvek na formuláři.

Na začátku zdrojového kódu importujeme jmenný prostor QTOControlLib a QTOLibrary, dále deklarujeme proměnné.

```
...
using QTOLibrary;
using QTOControlLib;
...
private AxQTOControlLib.AxQTControl axQTControl1;
private MovieInfo movieInfo = null;
private bool m_bManualFormResize = false;
```

Aplikační logikou se zde nebudeme zabývat a následovat boudou výpisy částí kódu, kde je zacházno s frameworkem QT. Načtení video souboru je prováděno v proceduře OpenMovie, ošetřeny jsou i chybové stavy.

```
private void OpenMovie(string url)
        {   string errMsg = "";
            try
            {
                axQTControl1.Sizing = QTSizingModeEnum.qtControlFi-
tsMovie;
                axQTControl1.URL = url;
                axQTControl1.Sizing = QTSizingModeEnum.qtMovieFi-
tsControl;
                addMovieEventListeners(axQTControl1.Movie);
                if (movieInfo != null)
                {
                    if (movieInfo.Created && movieInfo.Visible)
                    {
                        movieInfo.SetInfoMovie(axQTControl1.Movie);
                    }
                }
            }
            catch (COMException except)
            {
```

```
                errMsg="Chybovy kod: "+except.Error-
Code.ToString("X")+"\n";
                QTUtils qtU = new QTUtils();
                errMsg += "Chybovy kod QT: " +
                     qtU.QTErrorFromErrorCode(except.Error-
Code).ToString();
            }
            catch (Exception except)
            {   errMsg = except.ToString();
            }
            if (errMsg != "")
            {
                string msg = "Film nelze otevrit:\n\n";
                msg += url + "\n\n";
                msg += errMsg;
                MessageBox.Show(msg, "Chyba", MessageBoxBut-
tons.OK,                    MessageBoxIcon.Error);
            }
        }
```

Zavření video souboru provedeme vyprázdněním komponenty QT.

```
private void mnuClose_Click(object sender, EventArgs e)
        {
            if (axQTControl1.Movie == null) return;
            removeMovieEventListeners(axQTControl1.Movie);
            axQTControl1.URL = "";
            if (movieInfo != null)
            {
                if (movieInfo.Created && movieInfo.Visible)
                    movieInfo.Dispose();
            }
            axQTControl1.Hide();
        }
```

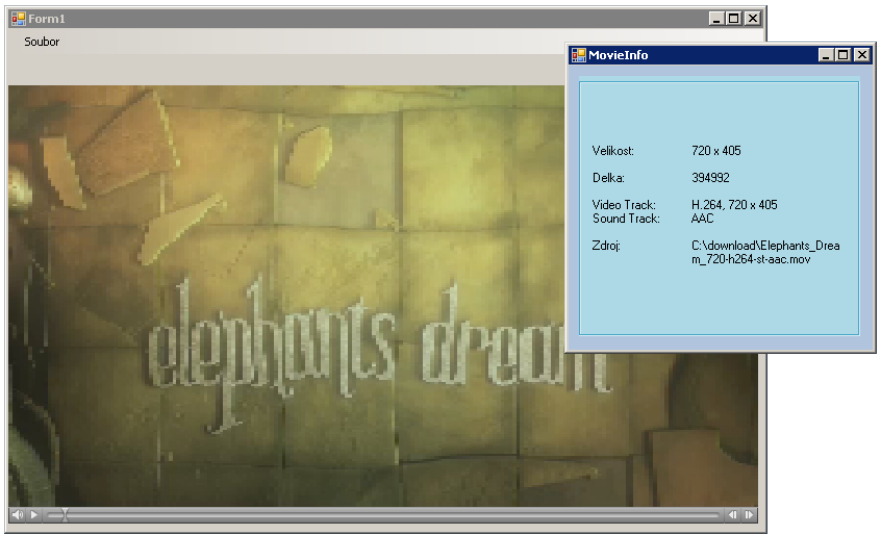
Proměnná movieInfo ukazje na další formulář, který obsahuje informace o přehrávaném souboru. Pro výpis informací voláme proceduru SetInfoMovie z námi vytvořené třídy MovieInfo.

```
private void getInfo()
        {
            if (axQTControl1.URL != "")
            {
                if (movieInfo == null)
                {
                    movieInfo = new MovieInfo();
                }
                else if (movieInfo.Created == false)
                {
                    movieInfo = new MovieInfo();
                }
                movieInfo.SetInfoMovie(axQTControl1.Movie);
                movieInfo.Show();
                movieInfo.BringToFront();
            }
        }
```

Přehrávaný soubor je také možné uložit, respektive exportovat s podrobným nastavením do kontejneru mov. Funkce exportu je plně implementována v komponentě QuickTime Control.

```
public void Export()
```

```
        {   if (axQTControl1.Movie == null) return;
            try
            {
                QTQuickTime qt = axQTControl1.QuickTime;
                if (qt.Exporters.Count == 0) qt.Exporters.Add();
                QTExporter ex = qt.Exporters[1];
                ex.TypeName = "QuickTime Movie";
                ex.SetDataSource(axQTControl1.Movie);
                ex.ShowSettingsDialog();
                saveFileDialog1.FileName = "exportovano.mov";
                if (saveFileDialog1.ShowDialog() == DialogRe-
sult.OK)
                {
                    ex.DestinationFileName = saveFileDialog1.Fi-
leName;
                    ex.ShowProgressDialog = true;
                    Cursor.Current = Cursors.WaitCursor;
                    ex.BeginExport();
                    Cursor.Current = Cursors.Arrow;
                }
            }
            catch (COMException except)
            {
                if (except.ErrorCode != -2147156096)
                     MessageBox.Show("Chybovy kod: " +
                     except.ErrorCode.ToString("X"), "Chyba expor-
tu");
            }
            catch (Exception except)
            {
                MessageBox.Show(except.ToString(), "Chyba expor-
tu");
            }
        }
```



Obr. 5.1 Ukázka rozhraní videopřehrávače spolu s výpisem informací o video souboru

Funkčnost je ověřena na souborech MOV, ve kterých byl uložen audiovizuální klip, soubory byly v různých rozlišení. Kompresní formát souboru [23] je H.264 pro video a komprese zvuku AAC.

5.2 Jednoduchá aplikace DirectShow

Jednoduchý video přehrávač využívající framework DirectShow. DirectShow je součástí komplexního rozhraní DirectX. Potřebné knihovny a dokumentaci nalezneme ve vývojovém balíku DirectX SDK. Ukázkový program má grafické uživatelské rozhraní a je napsán v jazyce C#, čímž lze využít možnosti moderního programovatelného rozhraní Microsoft .NET Framework a tedy možnost programovat objektově. Objektově orientované programování je přívětivější a může se mu věnovat i programátor-amatér.

Aplikaci jsem programoval ve vývojovém prostředí Microsoft Visual C# 2008 Express Edition. Grafickou podobu rozhraní jsem vytvořil ve vektorovém grafickém editoru Inkscape. Pomocné grafické úpravy jsem porváděl v programu GIMP.

Funkčnost přehrávače je obstarána instancí třídy Video ze jmenného prostoru Microsoft.DirectX.AudioVideoPlayback. Nejprve je nutné jmenný prostor vložit do programu skrze referenci na příslušnou knihovnu dll. DirectX SDK se při instalaci bezproblémově zapíše do mezipaměti sestavení (Global Assembly Cache) a je k nalezení mezi komponentami rozhraní .NET.

Na částech zdrojového kódu můžeme vidět manipulaci s třídou Video. Nejprve musíme importovat jmený prostor DirectX. Dále deklarujeme datové položky, proměnné a události potřebné ve vlastní obslužné třídě videoprehravac.

```
...
using Microsoft.DirectX.AudioVideoPlayback;

namespace OOP
{    public class Videoprehravac
    {   private Video video;
        private Form Formular;
        private Form Formular2;
        private bool Pozastaveno = false;
        public delegate void KonecPrehravaniDelegat();
        public event KonecPrehravaniDelegat KonecPrehravani;
...
```

Na další části je přetížený instanční konstruktor třídy Videoprehravac a metoda iniciující zahájení přehrávání videosouboru. Následují ostatní obslužné metody pro práci s video souborem. Ukázkový přehrávač umí pouze načíst soubor, spustit jej, pozastavit a ukončit. Přehrávané video je vykresleno na zvláštní formulář a může být zobrazeno jak v okně, tak v celo obrazovkovém režimu. Pokud je video spuštěno v okně, je viditelný ukazatel průběhu přehrávání.

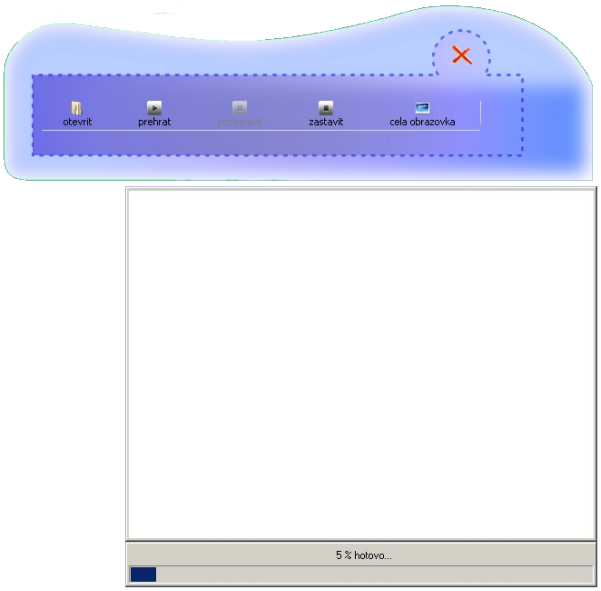
```
...
        public Videoprehravac(string Soubor, bool ZacitPrehravat)
            : this(Soubor)
        {
            if (ZacitPrehravat) this.Prehrat();   }

        public void Prehrat()
        {            video.Play();        }
        public void Pozastavit()
        {            video.Pause();        }
```

```
        public void Zastavit()
        {
            this.UkazatelPrubehu.Value =
this.UkazatelPrubehu.Minimum;
            this.Popisek.Text = this.UkazatelPrubehu.Value + " %
hotovo...";
            video.Stop();        }

        public void Previnout()
        {            video.CurrentPosition = 0.0d;        }

        public void UkoncitPrehravani()
        {
            this.Casovac.Enabled = false;
            Formular.Close();
            Formular2.Close();
            video.Dispose();        }
...
        public bool FullScreen
        {   get { return video.Fullscreen; }
            set { video.Fullscreen = value; }        }
...
```



Obr. 5-2 Ukázka grafické podoby přehrávače

Aplikační logika přehrávače již není z pohledu tématu důležitá. Následuje část obslužného kódu při stisku tlačítka přehrát.

```
private void tSButton2_Click(object sender, EventArgs e)
        {
            if (video == null)
                video = new OOP.Videoprehravac(Soubor);
            video.KonecPrehravani += new
OOP.Videoprehravac.KonecPrehravaniDelegat(video_KonecPrehravani);
            if (!ZahajeniPrehravani)
            {
                if (tSButton5.Checked)
                    video.FullScreen = true;
                video.Prehrat();
                tSButton2.Enabled = false;
                tSButton3.Enabled = true;
                ZahajeniPrehravani = true;            }
        }
```

5.3 Jednoduchá aplikace GStreamer

Následující zdrojový kód je součástí dokumentace pro framework Gstreamer a nalezneme ho v desaté kapitole [24]. Jedná se o opis originálního kódu a uveden je zde pro ucelení tématu. GStreamer je zástupce frameworku určeného primárně pro systém Linux.

V ukázce se jedná o jednoduchý přehrávač audio souborů spustitelný v příkazové řádce. Podporované soubory jsou pouze Ogg/Vorbis.

Potřebné zdrojové soubory GStreameru nalezneme například na oficiálních stránkách. Pokud se zaměříme na programování ve Windows, nesmíme opomenout fakt, že spolu s GStremerem budeme muset ještě doinstalovat další knihovny pocházející z prostředí Linuxu. Mezi potřebné knihovny patří GLib, libxml2, libintl a libiconv. Buď si dané knihovny stáhneme jednotlivě, nebo použijeme některý s balíčků knihoven, který obsahuje tyto knihovny v jednom instalačním souboru. Rozsáhlý balíček knihoven Linuxu je např. GNUWin32.

Výpis zdrojového kódu:

```
#include <gst/gst.h>
/*
 * Global objects are usually a bad thing. For the purpose of this
 * example, we will use them, however.
 */

GstElement *pipeline, *source, *parser, *decoder, *conv, *sink;

static gboolean
bus_call (GstBus     *bus,
      GstMessage *msg,
      gpointer    data)
{
  GMainLoop *loop = (GMainLoop *) data;

  switch (GST_MESSAGE_TYPE (msg)) {
    case GST_MESSAGE_EOS:
      g_print ("End-of-stream\n");
      g_main_loop_quit (loop);
      break;
    case GST_MESSAGE_ERROR: {
      gchar *debug;
      GError *err;

      gst_message_parse_error (msg, &err, &debug);
      g_free (debug);

      g_print ("Error: %s\n", err->message);
      g_error_free (err);

      g_main_loop_quit (loop);
      break;
    }
    default:
      break;
  }
```

```
  return TRUE;
}

static void
new_pad (GstElement *element,
     GstPad     *pad,
     gpointer    data)
{
  GstPad *sinkpad;
  /* We can now link this pad with the audio decoder */
  g_print ("Dynamic pad created, linking parser/decoder\n");

  sinkpad = gst_element_get_pad (decoder, "sink");
  gst_pad_link (pad, sinkpad);

  gst_object_unref (sinkpad);
}

int
main (int   argc,
      char *argv[])
{
  GMainLoop *loop;
  GstBus *bus;

  /* initialize GStreamer */
  gst_init (&argc, &argv);
  loop = g_main_loop_new (NULL, FALSE);

  /* check input arguments */
  if (argc != 2) {
    g_print ("Usage: %s <Ogg/Vorbis filename>\n", argv[0]);
    return -1;
  }

  /* create elements */
  pipeline = gst_pipeline_new ("audio-player");
  source = gst_element_factory_make ("filesrc", "file-source");
  parser = gst_element_factory_make ("oggdemux", "ogg-parser");
  decoder = gst_element_factory_make ("vorbisdec", "vorbis-decoder");
  conv = gst_element_factory_make ("audioconvert", "converter");
  sink = gst_element_factory_make ("alsasink", "alsa-output");
  if (!pipeline || !source || !parser || !decoder || !conv || !sink) {
    g_print ("One element could not be created\n");
    return -1;
  }

  /* set filename property on the file source. Also add a message
   * handler. */
  g_object_set (G_OBJECT (source), "location", argv[1], NULL);

  bus = gst_pipeline_get_bus (GST_PIPELINE (pipeline));
  gst_bus_add_watch (bus, bus_call, loop);
  gst_object_unref (bus);

  /* put all elements in a bin */
  gst_bin_add_many (GST_BIN (pipeline),
            source, parser, decoder, conv, sink, NULL);

  /* link together - note that we cannot link the parser and
   * decoder yet, becuse the parser uses dynamic pads. For that,
```

```
   * we set a pad-added signal handler. */
  gst_element_link (source, parser);
  gst_element_link_many (decoder, conv, sink, NULL);
  g_signal_connect (parser, "pad-added", G_CALLBACK (new_pad),
NULL);

  /* Now set to playing and iterate. */
  g_print ("Setting to PLAYING\n");
  gst_element_set_state (pipeline, GST_STATE_PLAYING);
  g_print ("Running\n");
  g_main_loop_run (loop);

  /* clean up nicely */
  g_print ("Returned, stopping playback\n");
  gst_element_set_state (pipeline, GST_STATE_NULL);
  g_print ("Deleting pipeline\n");
  gst_object_unref (GST_OBJECT (pipeline));

  return 0;
}
```

Závěr

Původně se práce měla podle zadání soustředit pouze na moderní multimediální framework. Při prohledávání informačních zdrojů jsem však narazil na problém, že spousta odborných článků nemá o pojmu multimediální framework shodné informace. Některé laické články dokonce zaměňují pojmy framework a kontejner. Podobná informativní nesrovnalost platí i pro kompresní formáty. Proto jsem se rozhodl začlenit kapitolu věnovanou kompresi digitálního videa a multimediálním kontejnerům. Ucelil jsem tak pojmy, které spolu v praxi úzce souvisí. Žádný framework se neobejde bez multimediálního kontejneru, kontejner obsahuje vždy multimediální data v komprimované podobě a data nesou ve většině případů audio nebo video. Video je pak pochopitelně ve formátu SDTV nebo HDTV.

Podle mého názoru zůstanou multimediální frameworky zahaleny tajemstvím. Částečně je to dáno filozofií, která se snaží výpočetní techniku přiblížit, co nejširší skupině lidí. Tento způsob prosazuje snadné uživatelské ovládání bez nutnosti znát provozní detaily. Stejně tak je tomu i u frameworku. Programování přes uživatelsky přívětivé rozhraní nenutí uživatele, aby znal základní stavbu daného programovacího nástroje. Více než framework se do popředí dostávají zmínky o multimediálních kontejnerech a to konkrétně o formátu Matroska, MPEG TS a MP4. Mezi formáty pak vévodí H.264, jeho open source verze x.264, dále VC-1 a MPEG-2. To vše se děje díky velkému zájmu o formát HDTV. Co je opravdu nejpopulárnější v oblasti multimedií zjistíme nejlépe při pohledu na některou výměnnou síť peer-to-peer. Ačkoliv se jedná o nelegální činnost, distribuce pirátských filmů ukazuje směr technologického vývoje.

Přestože jsou multimediální frameworky tajemným pojmem, nezapadnou do zapomnění. Nevědomky je totiž používáme všichni a to především v oblasti streamového videa. Na Internetu se běžně setkáme s formáty QuickTime a Helix DNA. Uživatelé linuxových distribucí hojně využívají služby GStreameru a přehrávání multimedií na vyspělých mobilních telefonech by se bez Multi Media Framework jen těžko obešlo.

V praktické části jsem ukázal práci s frameworky pomocí jednoduchých videopřehrávačů. Použitý programovací jazyk C# jsem využil pro jeho menší náročnost na programátorské schopnosti. Zkušenějším programátorům by zřejmě více vyhovovala práce v C++. V jazyce C++ jsme již nuceni znát detailně strukturu frameworku a vědět o možnostech, které jsou nám pomocí základních příkazů nabízeny.

Příloha - Zdrojové kódy

Jednoduchá aplikace QuickTime

Výpis primárního souboru Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using QTOLibrary;
using QTOControlLib;
using System.Runtime.InteropServices;

namespace videoprehravac_QT
{
    public partial class Form1 : Form
    {
        private MovieInfo movieInfo = null;
        private bool m_bManualFormResize = false;

        public Form1()
        {
            InitializeComponent();
        }

        private void mnuOpen_Click(object sender, EventArgs e)
        {
            string filePath = GetFileName();
            if (filePath != "") OpenMovie(filePath);
        }

        private void OpenMovie(string url)
        {
            string errMsg = "";
            try
            {
                axQTControl1.Sizing = QTSizingModeEnum.qtControlFitsMovie;
                axQTControl1.URL = url;
                axQTControl1.Sizing = QTSizingModeEnum.qtMovieFitsControl;
                addMovieEventListeners(axQTControl1.Movie);
                Form1.ActiveForm.Width = axQTControl1.Width;
                Form1.ActiveForm.Height =Form1.ActiveForm.MainMenuStrip.Height
                   + axQTControl1.Height + 30;
                if (movieInfo != null)
                {
                    if (movieInfo.Created && movieInfo.Visible)
                    {
                        movieInfo.SetInfoMovie(axQTControl1.Movie);
                    }
                }
            }
            catch (COMException except)
            {
                errMsg = "Chybovy kod: " + except.ErrorCode.ToString("X") +
                   "\n";
                QTUtils qtU = new QTUtils();
               errMsg += "Chybovy kod QT: " + qtU.QTErrorFromErrorCode
                   (except.ErrorCode).ToString();
            }
            catch (Exception except)
            {
```

```
                errMsg = except.ToString();
            }

            if (errMsg != "")
            {
                string msg = "Film nelze otevrit:\n\n";
                msg += url + "\n\n";
                msg += errMsg;
                MessageBox.Show(msg, "Chyba", MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
            }
        }

        private string GetFileName()
        {
            if (openFileDialog1.ShowDialog() == DialogResult.OK)
                return openFileDialog1.FileName;
            else
                return "";
        }

        private void mnuOpenURL_Click(object sender, EventArgs e)
        {
            InputForm input = new InputForm();
            DialogResult dialogResult = input.DisplayInputBox();
            if (dialogResult == DialogResult.OK)
OpenMovie(input.GetTextResult());
        }

        private void mnuClose_Click(object sender, EventArgs e)
        {
            if (axQTControl1.Movie == null) return;
            removeMovieEventListeners(axQTControl1.Movie);
            axQTControl1.URL = "";
            if (movieInfo != null)
            {
                if (movieInfo.Created && movieInfo.Visible)
                    movieInfo.Dispose();
            }
            axQTControl1.Hide();
        }

        private void mnuExport_Click(object sender, EventArgs e)
        {
            Export();
        }

        private void mnuFullscreen_Click(object sender, EventArgs e)
        {
            if (axQTControl1.URL != "")
            {
                axQTControl1.FullScreen = true;
            }
        }

        private void mnuGetInfo_Click(object sender, EventArgs e)
        {
            getInfo();
        }

        private void getInfo()
        {
            if (axQTControl1.URL != "")
            {
                if (movieInfo == null)
                {
```

```
            movieInfo = new MovieInfo();
        }
        else if (movieInfo.Created == false)
        {
            movieInfo = new MovieInfo();
        }
        movieInfo.SetInfoMovie(axQTControl1.Movie);
        movieInfo.Show();
        movieInfo.BringToFront();
    }
}

private void mnuExit_Click(object sender, EventArgs e)
{
    Application.Exit();
}

private void axQTControl1_QTEvent(object sender,
    AxQTOControlLib._IQTControlEvents_QTEventEvent e)
{
    switch (e.eventID)
    {
        case (int)QTEventIDsEnum.qtEventShowStatusStringRequest:
            {
                string msg = e.eventObject.GetParam
                 (QTEventObjectParametersEnum.
                 .qtEventParamStatusString).ToString();
                Console.WriteLine("qtEventShowStatusStringRequest:
                  " + msg + "\n");
            }
            break;
        case (int)QTEventIDsEnum.qtEventRateWillChange:
            {
                string rate= e.eventObject.GetParam
                 (QTEventObjectParametersEnum.
                 .qtEventParamMovieRate).ToString();
                Console.WriteLine("RateWillChange to: " + rate + "\n");
            }
            break;
        case (int)QTEventIDsEnum.qtEventTimeWillChange:
            {
                string time = e.eventObject.GetParam
                 (QTEventObjectParametersEnum.
                 .qtEventParamMovieTime).ToString();
                Console.WriteLine("TimeWillChanged to:" + time +"\n");
            }
            break;
        case (int)QTEventIDsEnum.qtEventAudioVolumeDidChange:
            {
                string vol = e.eventObject.GetParam
                 (QTEventObjectParametersEnum.
                 .qtEventParamAudioVolume).ToString();
                Console.WriteLine("AudioVolumeDidChange to:"+vol+"\n");
            }
            break;
    }
}

private void axQTControl1_SizeChangedEventHandler
   (object sender,AxQTOControlLib._IQTControlEvents_SizeChangedEvent e)
{
    if(m_bManualFormResize) return;
    this.ClientSize=axQTControl1.Size;
}
```

```
        private void axQTControl1_StatusUpdateEventHandler(object sender,
AxQTOControlLib._IQTControlEvents_StatusUpdateEvent e)
        {
            switch (e.statusCodeType)
            {
                case (int)QTStatusCodeTypesEnum.qtStatusCodeTypeControl:
                    {
                        switch (e.statusCode)
                        {
                            case(int)QTStatusCodesEnum.qtStatusFullScreenBegin:
                                this.Hide();
                                break;
                            case(int)QTStatusCodesEnum.qtStatusFullScreenEnd:
                                axQTControl1.SetScale(1);
                                this.Show();
                                break;
                        }
                    }
                    break;
            }
        }


        private void Form1_Resize(object sender, EventArgs e)
        {
            m_bManualFormResize=true;
            axQTControl1.Size=this.ClientSize;
            m_bManualFormResize=false;
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            axQTControl1.ErrorHandling=(int)QTErrorHandlingOptionsEnum.
            .qtErrorHandlingRaiseException;
            axQTControl1.Sizing=QTSizingModeEnum.qtControlFitsMovie;
        }

        private void addMovieEventListeners(QTMovie myMovie)
        {
            if (myMovie == null) return;

            myMovie.EventListeners.Add(QTEventClassesEnum.qtEventClassApplicati
onRequest, QTEventIDsEnum.qtEventShowStatusStringRequest, 0, null);
            myMovie.EventListeners.Add(QTEventClassesEnum.qtEventClassApplicati
onRequest, QTEventIDsEnum.qtEventRateWillChange, 0, null);
            myMovie.EventListeners.Add(QTEventClassesEnum.qtEventClassApplicati
onRequest, QTEventIDsEnum.qtEventTimeWillChange, 0, null);
            myMovie.EventListeners.Add(QTEventClassesEnum.qtEventClassApplicati
onRequest, QTEventIDsEnum.qtEventAudioVolumeDidChange, 0, null);
        }

        public void removeMovieEventListeners(QTMovie myMovie)
        {
            if (myMovie != null) myMovie.EventListeners.RemoveAll();
        }


        public void Export()
        {
            if (axQTControl1.Movie == null) return;

            try
            {
                QTQuickTime qt = axQTControl1.QuickTime;
                if (qt.Exporters.Count == 0) qt.Exporters.Add();
                QTExporter ex = qt.Exporters[1];
```

```
                ex.TypeName = "QuickTime Movie";

                ex.SetDataSource(axQTControl1.Movie);
                ex.ShowSettingsDialog();
                saveFileDialog1.FileName = "exportovano.mov";
                if (saveFileDialog1.ShowDialog() == DialogResult.OK)
                {
                    ex.DestinationFileName = saveFileDialog1.FileName;
                    ex.ShowProgressDialog = true;
                    Cursor.Current = Cursors.WaitCursor;
                    ex.BeginExport();
                    Cursor.Current = Cursors.Arrow;
                }
            }
            catch (COMException except)
            {
                if (except.ErrorCode != -2147156096) MessageBox.Show("Chybovy
                   kod: "+ except.ErrorCode.ToString("X"), "Chyba exportu");
            }
            catch (Exception except)
            {
                MessageBox.Show(except.ToString(), "Chyba exportu");
            }

        }
    }
}
```

Jednoduchá aplikace DirectShow

Výpis třídy videoprehravac.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.DirectX.AudioVideoPlayback;
using System.Windows.Forms;
using System.Drawing;

namespace OOP
{
    public class Videoprehravac
    {
        private Video video;
        private Form Formular;
        private Form Formular2;
        private ProgressBar UkazatelPrubehu;
        private Label Popisek;
        private Timer Casovac;
        private int SirkaVidea, VyskaVidea;
        private bool Pozastaveno = false;
        public delegate void KonecPrehravaniDelegat();
        public event KonecPrehravaniDelegat KonecPrehravani;

        public Videoprehravac(string Soubor)
        {
            video = Video.FromFile(Soubor);
            video.Ending += new EventHandler(video_Ending);
            Formular = new Form();
            video.Owner = Formular;
            Formular.MinimizeBox = false;
            Formular.ControlBox = false;
            Formular.ClientSize = video.DefaultSize;
            Formular.StartPosition = FormStartPosition.Manual;
            Formular.Location =
```

```
            new Point((Screen.PrimaryScreen.Bounds.Width - Formular.Width)
                /2,(Screen.PrimaryScreen.Bounds.Height-Formular.Height)/2);
        Formular.SizeChanged += new EventHandler(Formular_SizeChanged);
        Formular.KeyDown += new KeyEventHandler(Formular_KeyDown);
        Formular2 = new Form();
        Formular2.ShowInTaskbar = false;
        Formular2.Size = new Size(Formular.Width, 50);
        Formular2.ControlBox = false;
        Formular2.MinimizeBox = false;
        Formular2.StartPosition = FormStartPosition.Manual;
        Formular2.Location = new Point(Formular.Location.X,
                Formular.Location.Y + Formular.Height);
        Popisek = new Label();
        Popisek.Size = new Size(Formular2.Width, 20);
        Popisek.Location = new Point(0, 0);
        Popisek.TextAlign = ContentAlignment.MiddleCenter;
        UkazatelPrubehu = new ProgressBar();
        UkazatelPrubehu.Maximum = (int)video.Duration;
        UkazatelPrubehu.Height = 20;
        UkazatelPrubehu.Dock = DockStyle.Bottom;
        Formular2.Controls.Add(Popisek);
        Formular2.Controls.Add(UkazatelPrubehu);
        Formular.Show();
        Formular2.Show();
        Casovac = new Timer();
        Casovac.Enabled = true;
        Casovac.Interval = 100;
        Casovac.Tick += new EventHandler(Casovac_Tick);
        this.SirkaVidea = video.DefaultSize.Width;
        this.VyskaVidea = video.DefaultSize.Height;
    }

    public Videoprehravac(string Soubor, bool ZacitPrehravat)
        : this(Soubor)
    {
        if (ZacitPrehravat) this.Prehrat();
    }

    public void Prehrat()
    {
        video.Play();
    }

    public void Pozastavit()
    {
        video.Pause();
    }

    public void Zastavit()
    {
        this.UkazatelPrubehu.Value = this.UkazatelPrubehu.Minimum;
        this.Popisek.Text = this.UkazatelPrubehu.Value + " % hotovo...";
        video.Stop();
    }

    public void Previnout()
    {
        video.CurrentPosition = 0.0d;
    }

    public void UkoncitPrehravani()
    {
        this.Casovac.Enabled = false;
        Formular.Close();
        Formular2.Close();
        video.Dispose();
```

```
        }

        public int Sirka
        {
            get { return this.SirkaVidea; }
        }

        public int Vyska
        {
            get { return this.VyskaVidea; }
        }

        public bool FullScreen
        {
            get { return video.Fullscreen; }
            set { video.Fullscreen = value; }
        }

        private void Formular_SizeChanged(object sender, EventArgs e)
        {
            Formular2.Size = new Size(Formular.Width, 50);
            Formular2.Location = new Point(Formular.Location.X,
             Formular.Location.Y + Formular.Height);
            Popisek.Size = new Size(Formular2.Width, 20);
        }

        void Formular_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Escape)
                video.Fullscreen = false;
            else if (e.Control && e.KeyCode == Keys.P)
            {
                if (!Pozastaveno)
                {
                    this.Pozastavit();
                    Pozastaveno = true;
                }
            }
        }

        void Casovac_Tick(object sender, EventArgs e)
        {
            UkazatelPrubehu.Value = (int)video.CurrentPosition;
            Popisek.Text = (int)((video.CurrentPosition / video.Duration) *
            *100) + " % hotovo...";
        }

        private void video_Ending(object sender, EventArgs e)
        {
            KonecPrehravani();
            Previnout();
        }

    }
}
```

Seznam ilustrací

Použitá literatura a citace

[1] Jahoda, R. *Komerční formáty videa a TV*. Zpráva, 14.3.2002. [online] <www.tvfreak.cz>

[2] Jahoda, R. *Formáty obrazu videa*. Zpráva, 30.10.2001. [online] <www.tvfreak.cz>

[3] NHK (Japan Broadcasting Corporation) Super Hi-Vision, *Features of Super Hi-Vision*. 2005 [online] < www.nhk.or.jp/digital/en/super_hi>

[4] Tektronix, Inc. *New video compression standards: meeting the test challenges*. Technická zpráva, 2004. [online] [2AW_18398_1.pdf] www.tektronix.com/video

[5] Richardson I. E *H.264 and MPEG-4 Video Compression*, 2003 ISBN 0-470-84837-5 [online] [www.rgu.ac.uk/files/h264_interpred.pdf] ISBN 0-470-84837-5

[6] Líška D. *Digitální terestriální televize DVB-T: Technické minimum - MPEG 2*. Článek, 22.2. 2002. [online] < www.digitalnitelevize.cz/magazin/dvb-t>

[7] Jahoda, R. *Kodeky tajemství zbavené*. Zpráva, 13.10.2005. [online] <www.tvfreak.cz>

[8] Wikipedia – Internetová encyklopedie, Multimedia Framework. Články s licencí GNU <en.wikipedia.org/wiki/Multimedia_framework>

[9] Apple Inc., *QuickTime Overview*, 2005. Elektronický průvodce [online] <developer.apple.com/documentation/QuickTime/RM/Fundamentals/QTOverview>

[10] Apple Inc., *QuickTime Technologies*, Zpráva [online] <www.apple.com/quicktime/technologies/>

[11] Taymans, Baker, Wingo, Bultje, Kost GStreamer Application Development Manual. [online] <gstreamer.freedesktop.org/data/doc/gstreamer/head/manual>

[12] Symbian Software Ltd. *Online guide for Symbian OS v9.2*, 2006 [online] <www.symbian.com/developer/techlib/v9.2docs/doc%5Fsource/guide/>

[13] Jahoda, R. *Kontejner není kontejner*. Zpráva 10.5.2005. [online] <www.tvfreak.cz>

[14] Microsoft Co. *ASF Specification, rev. 01.20.03*. 12/2004. [online] <msdn2.microsoft.com/en-us/library/bb643323.aspx>

[15] CoreCodec Inc., *Matroska Technical specification*. [online] <www.matroska.org/technical/index.html>

[16] ITU-T Study Group 16, *ITU-T Recommendation H.264 (03/2005)*, 2005 [online] [T-REC-H.264-200503-I!!PDF-E.pdf] < www.itu.int/rec/T-REC-H.264/e>

[17] Jech V. *Digitální video, Prokládání, Kompresní formáty*. Internetové články. [online] < jech.webz.cz>

[18] Buriánek J. *Nové standardy pro komprimaci videa*, Pixel 105-Pixel 106, odborný časopis. ATLANTIDA Publishing s.r.o. ISSN: 1211-5401

Použitá literatura - praktická část

[19] Apple Computers Inc. *QuickTime Guide forWindows*. leden 2006. Elektronická publikace. [QTforWindows.pdf] <developer.apple.com/documentation/QuickTime/QuickTimeforWindows-date.html>

[20] Apple Computers Inc. *QuickTime Movie Basics*. leden 2006. Elektronická publikace. [MTEditing.pdf] <developer.apple.com/documentation/QuickTime/MovieBasics-date.html>

[21] Apple Computers Inc. *Windows API Reference for QuickTime*. květen 2006. Elektronická publikace. [QTRef_Windowsg.pdf] <developer.apple.com/reference/QuickTime/idxQuickTimeforWindows-date.html>

[22] Apple Computers Inc. *QuickTime for Windows*. 2003. ukázkové kódy. [online] <developer.apple.com/samplecode/QuickTime/idxQuickTimeforWindows-date.htm>

[23] Orange Open Movie Team *Elephants Dream*. Multimediální soubor [Elephants_Dream_1024-h264-st-aac.mov] < orange.blender.org/download>

[24] Taymans, Baker, Wingo, Bultje, Kost *GStreamer Application Development Manual (0.10.17.1)*. 2007. [online] <gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/html/index.html>

[25] Hanák, Ján *C# praktické příklady*. Grada Publishing, 2006, ISBN 80-247-0988-0